

Network Representation Learning in Social Media

by

Suhang Wang

A Dissertation Presented in Partial Fulfillment  
of the Requirement for the Degree  
Doctor of Philosophy

Approved July 2018 by the  
Graduate Supervisory Committee:

Huan Liu, Chair  
Charu Aggarwal  
Arunabha Sen  
Hanghang Tong

ARIZONA STATE UNIVERSITY

August 2018

## ABSTRACT

The popularity of social media has generated abundant large-scale social networks, which advances research on network analytics. Good representations of nodes in a network can facilitate many network mining tasks. The goal of network representation learning (network embedding) is to learn low-dimensional vector representations of social network nodes that capture certain properties of the networks. With the learned node representations, machine learning and data mining algorithms can be applied for network mining tasks such as link prediction and node classification. Because of its ability to learn good node representations, network representation learning is attracting increasing attention and various network embedding algorithms are proposed.

Despite the success of these network embedding methods, the majority of them are dedicated to static plain networks, i.e., networks with fixed nodes and links only; while in social media, networks can present in various formats, such as attributed networks, signed networks, dynamic networks and heterogeneous networks. These social networks contain abundant rich information to alleviate the network sparsity problem and can help learn a better network representation; while plain network embedding approaches cannot tackle such networks. For example, signed social networks can have both positive and negative links. Recent study on signed networks shows that negative links have added value in addition to positive links for many tasks such as link prediction and node classification. However, the existence of negative links challenges the principles used for plain network embedding. Thus, it is important to study signed network embedding. Furthermore, social networks can be dynamic, where new nodes and links can be introduced anytime. Dynamic networks can reveal the concept drift of a user and require efficiently updating the representation when new links or users are introduced. However, static network embedding algorithms

cannot deal with dynamic networks. Therefore, it is important and challenging to propose novel algorithms for tackling different types of social networks.

In this dissertation, we investigate network representation learning in social media. In particular, we study representative social networks, which includes attributed network, signed networks, dynamic networks and document networks. We propose novel frameworks to tackle the challenges of these networks and learn representations that not only capture the network structure but also the unique properties of these social networks.

## DEDICATION

I dedicate my dissertation work to my loving parents, Yinlong Wang and Guoyin Lin, for supporting me to pursue my dream!

I also dedicate this dissertation to my wife, Chao Shao, and my lovely son Junchi Wang, for supporting me all the way! Without their help, encouragement and accompany, this journey would have not been possible.

## ACKNOWLEDGMENTS

This dissertation is impossible without the help from my advisor Dr. Huan Liu. I would like to thank him for giving me large freedom through my Ph.D. study to explore various research problems and his excellent advising skills with great patience and guidance, which makes my Ph.D. experience colorful, exciting and productive. I'm fortunate to have Dr. Liu as my Ph.D. advisor, from whom I learnt many abilities that can benefit all my life: how to write papers and give presentations, how to find and address challenging problems, and how to establish my career and see the big vision. Dr. Liu is not only a supervisor for research but also a life mentor to help me improve my personality, overcome difficulties and make better decisions. He's very kind and helpful to give his suggestions and offer help for many aspects in my life and he's generous to share his experience and knowledge to help me avoid detours in my life. Dr.Liu, I cannot thank you enough.

I would like to thank my committee members, Dr. Charu Aggarwal, Dr. Arunabha Sen, and Dr. Hanghang Tong, for helpful suggestions and insightful comments. Part of this dissertation was done when I was an intern of Dr. Charu Aggarwal. His broad interests and knowledge largely broaden my perspectives on research and greatly expand the boundaries of my research. I took the algorithm course from Dr. Arunabha Sen, which prepared me with solid technical background and benefited my Ph.D. research a lot. I also took a seminar course on network mining from Dr. Hanghang Tong. His insightful discussions and comments provided me new angles to rethink about my research on network representation learning.

I was lucky to work as interns in IBM Research and Clari with amazing colleagues and mentors: Charu Aggarwal, Wenchao Yu, Qing Wang and Kun-Lung Wu in IBM Research; Lei Tang, Xin Xu, Venkat Rangan and Andy Bryne from Clari. Because of you, my life became much easier in new environments; because of you, I enjoyed two

wonderful and productive summers; and because of you, I was able to contribute my knowledge to exciting projects. Thank you for everything.

During my Ph.D. study, my friends and colleagues provided me consistent support and encouragement and they deserve a special thank. I am thankful to my colleagues at the Data Mining and Machine Learning Lab: Jiliang Tang, Huiji Gao, Xia Hu, Pritam Gundecha, Fred Morstatter, Shamanth Kumar, Ali Abbasi, Reza Zafarani, Rob Trevino, Isaac Jones, Tahora H. Nazer, Suhas Ranganath, Jundong Li, Liang Wu, Ghazaleh Beigi, Kai Shu, Kewei Cheng, Vineeth Rakesh, Lu Cheng, Nur Shazwani Kamrudin, Ruo Cheng Guo and Kaize Ding. In particular, thanks to Jiliang Tang who helped me write my first paper and taught me how to do research; thanks to my long-term collaborators Jiliang Tang, Xia Hu, Yilin Wang and Suhas Ranganath and I will remember a lot of deadlines we tried to beat; thanks to Fred Morstatter, Rob Trevino, and Isaac Jones as my English teachers and I will remember any error you corrected and every new word you taught; thanks to Reza Zafarani and Jiliang from whom I learned my presentation skills. I am also lucky to have you as friends and colleagues in my life: Yilin Wang, Yang Li, Xuying Meng, Shuo Yang, Yao Ma and Wen Zhang.

Finally, I am deeply indebted to my dear mother and father for their love and strong support during my graduate study. I would like to thank my dear wife Chao Shao for her strong support through all these years to my study. How fortunate I am having her in my life! This dissertation is dedicated to them.

## TABLE OF CONTENTS

	Page
LIST OF TABLES .....	x
LIST OF FIGURES .....	xi
CHAPTER	
1 INTRODUCTION .....	1
1.1 Research Challenges .....	3
1.2 Contributions .....	4
1.3 Organization .....	4
2 FOUNDATIONS AND PRELIMINARIES .....	6
2.1 Plain Network Embedding .....	6
2.2 Alternative Options to Network Embedding .....	8
3 ATTRIBUTED NETWORK EMBEDDING .....	10
3.1 Modeling Attributes with Restricted Boltzmann Machine .....	11
3.2 Paired Restricted Boltzmann Machine for Attributed Network Em- bedding .....	12
3.3 Training pRBM .....	16
3.3.1 Representation Learning with pRBM .....	22
3.3.2 Time Complexity .....	23
3.4 Evaluating pRBM .....	23
3.4.1 Experimental Settings .....	24
3.4.2 Quality of Learned Representations .....	25
3.4.3 Robustness of pRBM to Small Data .....	28
3.4.4 Parameter Sensitivity .....	31
4 SIGNED NETWORK EMBEDDING .....	32
4.1 An Objective Function for Signed Network Embedding .....	33

CHAPTER	Page
4.2 The Proposed Framework SiNE .....	37
4.3 The Architecture of SiNE .....	37
4.4 Optimization of SiNE .....	39
4.5 Training SiNE .....	40
4.5.1 Time Complexity .....	42
4.6 Extending SiNE for Attributed Signed Social Network .....	42
4.7 Evaluating SiNE .....	45
4.7.1 Datasets .....	45
4.7.2 Analysis of the Signed Embedding .....	46
4.7.3 Signed Link Prediction in Signed Social Networks .....	48
4.7.4 Parameter Analysis .....	50
5 DYNAMIC NETWORK EMBEDDING .....	53
5.1 Problem Formulation .....	55
5.2 Dynamic Network Embedding .....	56
5.2.1 Static Network Embedding .....	56
5.2.2 Bayesian Dynamic Network Embedding .....	58
5.3 Parameter Inference .....	60
5.3.1 Update Rules .....	61
5.3.2 A Training Algorithm .....	64
5.3.3 Time Complexity Analysis .....	65
5.4 Experiments .....	65
5.4.1 Datasets .....	67
5.4.2 Compared Network Embedding Algorithms .....	68
5.4.3 Next Timestamp Link Prediction .....	69



CHAPTER	Page
5.4.4	Node Classification . . . . . 74
5.4.5	Discovering Temporal Concept Drift: A Case Study . . . . . 75
5.4.6	Parameter Analysis . . . . . 77
6	DOCUMENT NETWORK EMBEDDING . . . . . 79
6.1	Problem Statement . . . . . 80
6.2	The Proposed Framework . . . . . 81
6.2.1	Modeling Word-Word-Document Relations . . . . . 81
6.2.2	Modeling Document-Document Relations . . . . . 82
6.2.3	Modeling Document-Label Relations . . . . . 83
6.2.4	Linked Document Embedding . . . . . 84
6.3	Learning LDE . . . . . 84
6.3.1	Approximation by Negative Sampling . . . . . 85
6.3.2	Updating Rules . . . . . 87
6.3.3	Subsampling of Frequent Words . . . . . 88
6.3.4	A Learning Algorithm for LDE . . . . . 88
6.3.5	Time Complexity . . . . . 89
6.4	Experiments . . . . . 90
6.4.1	Datasets and Experimental Settings . . . . . 91
6.4.2	Performance Comparison . . . . . 92
6.4.3	Impact of Label Density . . . . . 98
6.4.4	Effects of Link Density . . . . . 99
6.4.5	Effects of Embedding Dimensions . . . . . 101
7	CONCLUSION AND FUTURE WORK . . . . . 103
7.1	Summary . . . . . 103

CHAPTER	Page
7.2 Future Work .....	105
REFERENCES .....	107
A Optimization of SiNE .....	115
A.1 Summary of Derivatives .....	118
BIOGRAPHICAL SKETCH .....	120

## LIST OF TABLES

Table	Page
3.1 Statistics of the Attributed Social Network Datasets. ....	24
3.2 Accuracy(%) Comparison on BlogCatalog and Flickr. ....	27
3.3 NMI Comparison on BlogCatalog and Flickr ....	27
3.4 Statistics of the Reduced Datasets ....	29
4.1 Average User Attributes Similarities in $\mathcal{P}$ and $\mathcal{N}$ ....	44
4.2 P-value of t-test Results ....	44
4.3 Statistics of the Signed Network Datasets ....	46
4.4 Average Distance Between Users and Their Friends and Foes. ....	47
4.5 AUC Comparison of Signed Link Prediction on Epinions and Slashdot .	49
4.6 F1 Comparison of Signed Link Prediction on Epinions and Slashdot ...	49
4.7 AUC of SiNE on Signed Link Prediction with Different Number of Layers $N$ ....	52
5.1 Statistics of the Datasets ....	68
5.2 Link Prediction Performance Comparison on Infectious in Terms of AUC.	72
5.3 Link Prediction Performance Comparison on Digg in Terms of AUC. ..	72
5.4 Link Prediction Performance Comparison on DBLP in Terms of AUC. .	72
5.5 Link Prediction on GPM in Terms of AUC. ....	73
5.6 Link Prediction on GPP in Terms of AUC. ....	73
5.7 Top 5 Authors Sorted by $\ \boldsymbol{\mu}_t - \boldsymbol{\mu}_{t-1}\ _2$ for Year 2003 ....	77
6.1 Statistics of the Datasets ....	92
6.2 Document Classification Performance Comparison on DBLP and Blog- Catalog ....	93
6.3 Effects of Label Density for LDE. ....	96
6.4 Effects of Link Density on LDE. ....	100

## LIST OF FIGURES

Figure	Page
3.1 An Illustrative Example of Attributed Network. ....	11
3.2 An Illustration of Restricted Boltzmann Machine. Figure (a) Is the Structure of an RBM. Figure (b) Is Simplified Representation of an RBM. ....	13
3.3 Paired Data .....	13
3.4 An Illustration of Paired Restricted Boltzmann Machine for Attributed Network Embedding .....	15
3.5 An Illustration of 3 Hidden-layer DBM .....	16
3.6 The Impact of the Size of Data on the Performance of Representation Learning Methods. ....	30
3.7 Sensitivity of pRBM to Dimensionality .....	31
4.1 Three Types of Triplets of Users. ....	34
4.2 Dealing with Special Case .....	35
4.3 Adding a Virtual Node. ....	36
4.4 An Illustration of the Architecture of SiNE with 2 Hidden Layers. ....	38
4.5 An Illustration of the Architecture of the Proposed SiNE for Attributed Signed Network. ....	44
4.6 A Case Study of Signed Network Embedding. ....	47
4.7 The Impact of Embedding Dimension $d$ on SiNE for Signed Link Prediction .....	51
4.8 The Impact of $\delta$ and $\delta_0$ on SiNE for Signed Link Prediction. ....	52
5.1 An Illustration of Dynamic Network .....	55
5.2 Graphical Illustration of DNE .....	58
5.3 Running Time Comparison on Digg and DBLP .....	74

Figure	Page
5.4 Node Classification Performance Comparison .....	76
5.5 Parameter Analysis of DNE on Link Prediction .....	78
5.6 Parameter Analysis of DNE on Node Classification .....	78
6.1 A Toy Example of Linked Documents. $\{d_1, d_2, \dots, d_5\}$ Are Documents; $\{w_1, w_2, \dots, w_8\}$ Are Words; $y_2$ Is the Label of $d_2$ and $y_5$ Is the Label of $d_5$ . .....	80
6.2 Relative Performance Improvement of LDE with Increasing Label In- formation. ....	98
6.3 Relative Performance Improvement of LDE with Increasing Link In- formation. ....	101
6.4 The Effects of Embedding Dimension on the Classification Performance of Document and Word Embedding. ....	102

## Chapter 1

### INTRODUCTION

The popularity of social media has generated abundant large-scale social networks, which advances the research on various network mining tasks such as node classification, node clustering and link prediction. Network representation learning, which aims at learning low-dimensional vector representations of nodes that capture certain properties of a network, can facilitate many network mining tasks. Generally, network representation learning has two advantages. First, it learns a good representation of nodes, enabling many machine learning and data mining algorithms designed for i.i.d data to be applied for network mining tasks. For example, with the learned representation, the node classification problem is reduced to a classical classification problem. Classifiers such as Support Vector Machines (SVMs) and deep neural networks can be used to perform classification. Second, the low-dimensional dense representation of nodes alleviates the network sparsity problem and the curse of dimensionality problem. Therefore, network representation learning or network embedding has attracted increasing attention [51, 55, 5, 64, 79, 89, 62, 91, 28, 91, 94]. However, the majority of existing network embedding algorithms are dedicated to static plain networks, i.e., networks with nodes and links; while in social media, networks can be present in various formats, such as attributed networks, signed networks, dynamic networks and heterogeneous networks.

These social networks contain abundant rich information to alleviate the network sparsity problem and can help learn a better network representation, while static plain network embedding approaches cannot tackle such networks. For example, attributed social networks are very popular because different types of information such as user

profile and preferences can be treated as node attributes. The attributes describe the preferences or properties of the node, which provide complementary information over network structure [40]. Thus, it is important to learn embedding that preserves both network information and node attributes. However, network embedding methods for plain networks don't take attributes into consideration. Similarly, signed social networks, which have both positive and negative links, are also pervasive. For example, in Epinions <sup>1</sup>, positive and negative links denote trust and distrust; and in Slashdot <sup>2</sup>, positive links mean friendships and negative links denote foes. In addition to existing signed social networks, many algorithms are proposed to construct signed networks from positive and negative interactions between users or documents [53]. Recent studies on signed network shows that negative links have added value over positive links for node classification [71] and link prediction [73]. Thus, it is important to learn embedding that encodes the semantic meanings of both positive and negative links. Furthermore, social networks can be dynamic, where existing users can change their preferences, add new friends and join new groups, or new users can join a social network and create new links anytime. Dynamic networks requires embedding algorithms to capture the concept drift of users and to efficiently update the representation of nodes when new links and new users are introduced. It is insufficient to directly apply plain network embedding algorithms on dynamic networks by ignoring the temporal information.

Therefore, in this dissertation, I investigate network representation learning in social media. In particular, I study representative social networks, which includes attributed network, signed networks, dynamic networks and document networks. I propose novel frameworks to tackle the challenges of these networks and learn repre-

---

<sup>1</sup><http://www.epinions.com/>

<sup>2</sup><https://slashdot.org/>

sentations that not only capture the network structure but also the unique properties of these networks.

## 1.1 Research Challenges

To learn representation for complex social networks, we are faced with several challenges:

- For attributed social networks, the attributes are usually presented as high-dimensional sparse binary vectors, with “1” meaning the existence of an attribute and “0” denoting missing attribute. How can we efficiently extract meaningful latent features from such binary vectors? And how can we learn network representations that encode both node attributes and network structure?
- For signed social networks, the existence of negative links challenges the principles used for unsigned network embedding. In unsigned social networks, homophily and social influence are applicable, which suggest that two linked nodes are likely to be similar to each other. However, in signed networks, two negatively linked nodes distrust each other or are foes. How can we tackle the semantic meanings of both positive and negative links? In addition, the underlying structure of the network is highly non-linear [52, 89]. The positive and negative links further increases its non-linearity. How can we design a model to tackle such highly non-linear networks?
- For dynamic networks, new links and new users can join any time and users’ preferences can gradually change over time. How can we capture the concept drift of nodes? How can we efficiently update/learn representations when new links/users are introduced anytime?



- For document networks, each node in the network is now represented as a document. One simple way is to use the bag-of-word representation, which converts the document network to an attributed network. However, such conversion disregards the order of words and ignores the semantic meaning of words. Therefore, how can we simultaneously capture the semantic meaning of words and documents? And how can we also take the links between two documents into consideration?

## 1.2 Contributions

The contributions of this dissertation are summarized as follows:

- Studying novel problems of network representation learning in social media such as signed network embedding and dynamic network embedding;
- Providing principled approaches to design network embedding algorithms guided by social theories for various types of networks in social media;
- Proposing novel frameworks to learn network representations from various social networks. The resulting representations preserve not only the network network structures, but also the unique properties of the social networks such as node attributes, signed links and dynamic patterns.
- Conducting experiments on real-world datasets to verify and demonstrate the effectiveness of the proposed frameworks.

## 1.3 Organization

The remainder of this dissertation is organized as follows. In Chapter 2, I review related works in network representation learning and its alternatives. In Chapter 3, I

investigate attributed network embedding. I first give details of the proposed framework paired Restricted Boltzmann Machine (pRBM), its training algorithms and time complexity. I then conduct experiments to evaluate the effectiveness of pRBM for attributed social network representation learning. In Chapter 4, I study signed network embedding. I first introduce the details of the proposed deep framework SiNE, whose design is guided by extended social balance theory. I then detail how to train SiNE, its time complexity and experimental results. In Chapter 5, I study dynamic network embedding. I first introduce a probabilistic framework DNE to dynamically learn representation in dynamic networks. I then give details of training with variational inference and reparameterization tricks and analyze time complexity. I further conduct experiments to understand DNE. In Chapter 6, I propose LDE for document network embedding. I first introduce how to capture the relations among words, documents and labels with LDE for representation learning. I then conduct experiments to evaluate LDE. I conclude the dissertation and point out broader impacts and promising research directions in Chapter 7.

### FOUNDATIONS AND PRELIMINARIES

In this section, we will briefly introduce the background about researches in plain network embedding and alternatives to network embedding.

#### 2.1 Plain Network Embedding

Network representation learning (or network embedding) aims at learning low-dimensional vector representations for nodes of a given network. It has been proven to be useful in many tasks of network analysis such as link prediction [51], community detection [63, 19], node classification [7, 92] and visualization [84, 77]. The heterogeneity in data representation, the sparsity of the network, and the varying degrees of various nodes, all play a significant role in making network mining tasks more challenging. To address these issues, network embedding encodes and represents each node in a unified low-dimensional space, which facilitates a better understanding of semantic relationships and enables the application of classical machine learning algorithms for network mining tasks [64]. For example, with the learned network representations, community detection problem is reduced to classical clustering algorithm and K-means can be applied.

Network embedding has attracted increasing attention in recent years and various network embedding algorithms are proposed [66, 5, 51, 64, 79, 62, 89, 93, 91, 29, 86, 65, 27]. For example, in [5], spectral analysis is performed on Laplacian matrix and the top- $k$  eigenvectors are used as the representations of network nodes. t-SNE proposed in [84] embeds the weighted network to low dimension for visualization by using stochastic neighbor embedding. SocDim [81] exploits network modularity to learn

the latent social dimensions as the node representation. The state-of-the-art network representation learning algorithms borrow the idea from word2vec [56] techniques. The essential idea is to first extract the node proximity from the network structure and then train the embedding to preserve the node proximity [64, 79, 28]. For example, DeepWalk [64] introduces the idea of Skip-gram [56], a word representation model in NLP, to learn node representations by extracting node proximity using random-walk sequences. Node2Vec [28] extends DeepWalk by introducing 2nd order random walk to extract node proximity [28]. LINE [79] exploits first order proximity and second order proximity to learn network embedding.

Deep learning is effective in representation learning, which has achieved great success in many domains such as computer vision [32], natural language processing [23] and speech recognition [26]. Therefore, more and more effort is dedicated to investigate deep learning models for network representation learning [89, 85]. For example, Wang et. al. [89] propose Structural Deep Network Embedding (SDNE), which utilizes deep networks to learn highly non-linear features. Graph attention network [85] adopts the attention mechanism to learn network representation, where attention mechanism is widely used in deep learning models [85]. Graph convolutional networks [43, 17] try to extend the concept of convolution in signal processing to learn network representation.

However, the majority of the aforementioned network representation learning algorithms are designed for plain networks, i.e., networks with fixed nodes and links; while in social media, networks can be present in different formats, such as attributed networks, signed networks, dynamic networks and document networks. Different type of social networks contains rich information that can help to alleviate the network sparsity problem. For example, negative links in signed social network usually mean distrust or foe relationship, which have different semantic meaning from positive links

and can be used to learn better network representation. However, existing plain network embedding algorithms cannot handle these rich information. Therefore, in this dissertation, we propose novel algorithms to tackle the challenges brought by these complex social networks, and learn representation that can simultaneously capture the network structure and unique properties of complex social networks.

## 2.2 Alternative Options to Network Embedding

There are several alternative options to network embedding. The goal of network embedding is to learn low dimensional vectors of nodes in a network such that the vectors capture certain properties of the network. These properties can be node structural proximity, node attributes, edge attributes and others, depending on the given networks. A high quality network embedding can preserve such properties and the representation can facilitate downstream network mining tasks such as link prediction, community detection and node classification. In other words, network embedding is also *feature learning* from networks to *facilitate downstream network mining tasks*. Based on this understanding, there are two alternatives to network embedding: (1) feature engineering; and (2) network mining algorithms.

Feature engineering [18] is the process of using domain knowledge of the data to create features that make machine learning algorithms work. If we can use the domain knowledge about the network or the downstream task to manually design and extract features from a network, these features may have as good quality as the features learned by network embedding algorithms. For example, for link prediction tasks, handcrafted features such as number of common neighbors between a pair of nodes are very indicative features for predicting links.

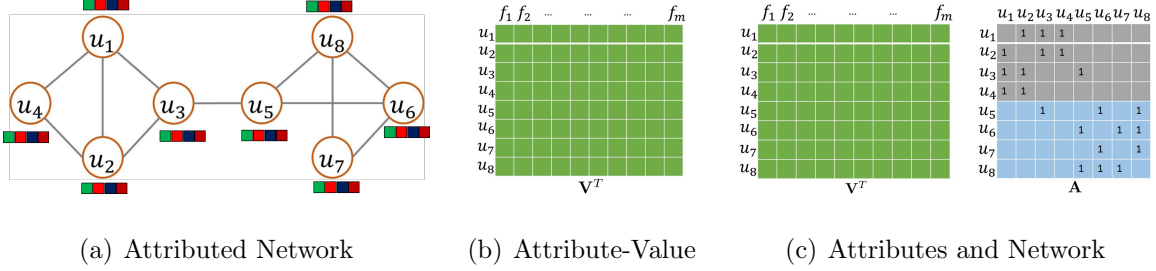
The ultimate goal of network embedding algorithms is to learn features that can facilitate downstream tasks. Thus, another choice is to design network mining algo-

rithms directly by utilizing the properties of a network without the explicit process of learning network embedding. For example, for community detection in signed networks, we want the majority of links within a group to be positive while the links between two groups to be negative. This can be used to design objective function for community detection without first learning network embedding and then perform clustering based on the learned embedding.

It is worth noting that these two alternatives can also be used together with network embedding. For example, the handcrafted features and the features learned by network embedding algorithms may contain complementary information, which can result in better network mining performances if they are concatenated together. In designing network mining tasks, the features learned by network embedding algorithms can be used as another source of input.

## ATTRIBUTED NETWORK EMBEDDING

In this chapter, we investigate attributed network embedding. An illustration of attributed network is shown in Figure 3.1. In addition to the network structure information, each node is associated with attributes. Attributed networks are very pervasive in social media. Various information can be treated as attributes of users in a social network. For example, a user’s profile such as gender, age, major, location and preferences can all be converted into binary vectors as attributes of the user. In addition to a user’s profile, social behaviors such as groups the user participated and genres of musics the user likes can also be treated as attributes of the user. Obviously, these attributes encode complementary information in addition to network structures, which has potential to alleviate the network sparsity problem in network representation learning. For example, as shown in Figure 3.1(a),  $u_4$  and  $u_5$  are not directly connected and thus it is difficult to know the similarity of these two users; while from node attributes, we have more information to understand their similarity. Therefore, it is important to learn an embedding that can simultaneously capture the network information and attribute information. In an attempt to learn attributed network embedding, we are faced with two challenges: (i) The attributes are usually represented as high-dimensional sparse binary vectors with “1” meaning the existence of this attribute and “0” denoting missing attributes. How to effectively model such high-dimensional sparse attribute information? and (ii) how to simultaneously capture both network structure and attribute information. To solve these two challenges, We proposed a novel framework paired restricted Boltzmann machine (pRBM), which



**Figure 3.1:** An Illustrative Example of Attributed Network.

will be introduced in detail next.

### 3.1 Modeling Attributes with Restricted Boltzmann Machine

To model the high-dimensional sparse attributes, restricted Boltzmann machine (RBM) [35] is a good fit because an RBM is an undirected graphical model that defines a probability distribution over a vector of observed, or visible, variables  $\mathbf{v} \in \{0, 1\}^m$  and a vector of latent, or hidden, variables  $\mathbf{h} \in \{0, 1\}^d$ . It is widely used for unsupervised representation learning and for pretraining deep learning models. Thus, RBM can learn meaningful features from such high-dimensional sparse attributes. Figure 3.2(a) gives a toy example of an RBM. In the figure, each node of the hidden layer is connected to each node in the visible layer, while there are no connections between hidden nodes or visible nodes. Figure 3.2(b) is a simplified representation of RBM, where the connection details between hidden layers and visible layers are simplified. In this work, we consider both  $\mathbf{v}$  and  $\mathbf{h}$  as binary vectors, *i.e.*, elements of  $\mathbf{v}$  and  $\mathbf{h}$  can only take the value of 0 or 1. An RBM defines a joint probability over  $\mathbf{v}$  and  $\mathbf{h}$  as,

$$P(\mathbf{v}, \mathbf{h}) = \exp(-E(\mathbf{v}, \mathbf{h}))/Z \quad (3.1)$$

where  $Z$  is a normalization constant, *i.e.*, the partition function, which is defined as  $Z = \sum_{\mathbf{v}} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))$  and  $E$  is an energy function given by

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{h}^T \mathbf{W} \mathbf{v} - \mathbf{b}^T \mathbf{h} - \mathbf{c}^T \mathbf{v} \quad (3.2)$$



where  $\mathbf{W} \in \mathbb{R}^{d \times m}$  is a matrix of pairwise weights between elements of  $\mathbf{v}$  and  $\mathbf{h}$  (see Figure 3.2(a)), while  $\mathbf{b} \in \mathbb{R}^{d \times 1}$  and  $\mathbf{c} \in \mathbb{R}^{m \times 1}$  are biases for the hidden and visible variables, respectively<sup>1</sup>. Since there are no explicit connections between hidden units in an RBM, given a randomly selected training instance  $\mathbf{v}$ , the hidden units are independent of each other

$$P(\mathbf{h}|\mathbf{v}) = \prod_{i=1}^d P(h_i|\mathbf{v}) \quad (3.3)$$

and the binary state,  $h_i$ ,  $i = 1, \dots, d$ , is set to 1 with conditional probability given as

$$P(h_i = 1|\mathbf{v}) = \sigma\left(\sum_{j=1}^m W_{ij}v_j + b_i\right) \quad (3.4)$$

where  $\sigma(\cdot)$  is the sigmoid function defined as  $\sigma(x) = (1 + \exp(-x))^{-1}$ . Similarly, given  $\mathbf{h}$ , the visible units are independent of each other, which result in

$$P(\mathbf{v}|\mathbf{h}) = \prod_{j=1}^m P(v_j|\mathbf{h}) \quad (3.5)$$

and the binary state,  $v_j$ ,  $j = 1, \dots, m$ , is set to 1 with conditional probability given as

$$P(v_j = 1|\mathbf{h}) = \sigma\left(\sum_{i=1}^d W_{ij}h_i + c_j\right) \quad (3.6)$$

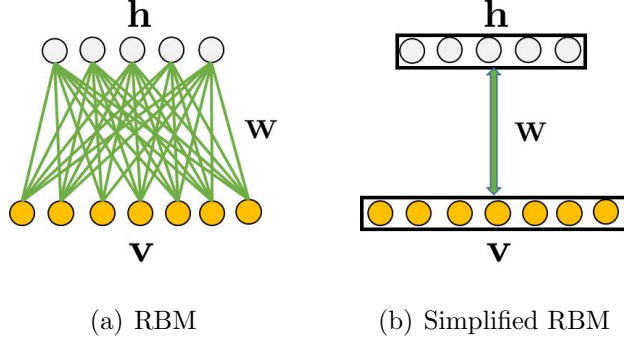
With the attributes  $\mathbf{v}$  as input, we can learn the latent representation of the user, i.e.,  $\mathbf{h}$ , with RBM.

### 3.2 Paired Restricted Boltzmann Machine for Attributed Network Embedding

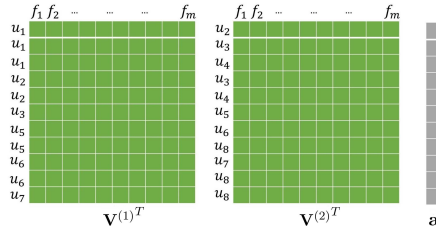
With RBM modeling the attributes, we are now going to incorporate the network information. The homophily and social influence theory are two social theories that can be used to design the model. The essential idea of homophily and social influence is that similar nodes may be more likely to attach to each other than dissimilar ones and two connected nodes become more similar. This suggests us to force the

---

<sup>1</sup>For simplicity, bias terms are not shown in Figure 3.2.



**Figure 3.2:** An Illustration of Restricted Boltzmann Machine. Figure (a) Is the Structure of an RBM. Figure (b) Is Simplified Representation of an RBM.



**Figure 3.3:** Paired Data

latent features of two linked users to be similar. Thus, we first extract pairs of users as  $\langle u_1, u_2 \rangle$  if  $u_1$  and  $u_2$  are linked, and represent attributed network from the edge perspective. For example, Figure 3.3 shows the pair representation of the attributed network in Figure 3.1, where each row in Figure 3.3 is a pair of nodes corresponding to an edge in Figure 3.1. The vector  $\mathbf{a}$  contains the link information and matrices  $\mathbf{V}^{(1)}$ ,  $\mathbf{V}^{(2)}$  contain the attributes of pairs of nodes. For the linked pair  $\langle u_1, u_2 \rangle$ , we use  $\mathbf{h}^{(1)} \in \{0, 1\}^{d \times 1}$  and  $\mathbf{h}^{(2)} \in \{0, 1\}^{d \times 1}$  to denote their latent feature representations. In order to model attribute networks from the edge perspective, we force latent feature representations of linked pairs of instances to be similar. To achieve this goal, we propose a novel representation learning algorithm paired Restricted Boltzmann Machine pRBM as demonstrated in Figure 3.4. pRBM is composed of two RBMs, which is designed for the pair representation. Since we assume that links are undirected, the two RBMs share the same parameters, such as  $\mathbf{W}$ ,  $\mathbf{b}$  and  $\mathbf{c}$ , which ensures that

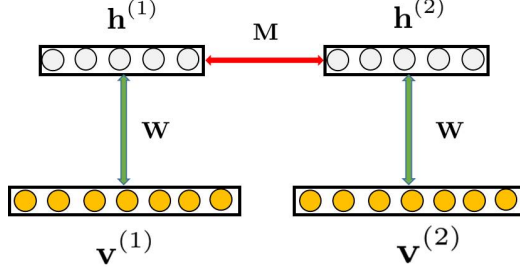
$\langle u_1, u_2 \rangle$  has the same effect to pRBM as  $\langle u_2, u_1 \rangle$ , *i.e.*, switching the order of a pair of nodes does not matter. Furthermore, by sharing the same parameters, we can reduce the number of parameters of pRBM greatly, which is significant for small datasets with high dimensionality of attributes because when datasets are small and the dimensionality of attributes is high, a complex model with large number of parameters cannot be well trained. As shown in Figure 3.4, hidden layers of the two RBMs are linked, which means that the feature representations  $\mathbf{h}^{(1)}$  and  $\mathbf{h}^{(2)}$  are fully connected. The connection between  $\mathbf{h}^{(1)}$  and  $\mathbf{h}^{(2)}$  allows interaction between them. This models the link between  $\langle u_1, u_2 \rangle$ . More specifically, as  $\langle u_1, u_2 \rangle$  are linked, it is likely that  $\langle u_1, u_2 \rangle$  share similar interests/topics, which implies that the similarity between  $\mathbf{h}^{(1)}$  and  $\mathbf{h}^{(2)}$  should be high. Thus, we learn a metric  $\mathbf{M} \in \mathbb{R}^{d \times d}$  to force latent feature representations of pairs of linked instances as  $\langle u_1, u_2 \rangle$  to be similar. Therefore, the joint probability of pRBM is defined as

$$P(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, a; \boldsymbol{\theta}) = \exp(-E(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, a))/Z \quad (3.7)$$

where  $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{b}, \mathbf{c}, \mathbf{M}\}$  is the parameter set and the energy function is defined as

$$\begin{aligned} E(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, a) = & -a(\mathbf{h}^{(1)})^T \mathbf{M} \mathbf{h}^{(2)} - (\mathbf{h}^{(1)})^T \mathbf{W} \mathbf{v}^{(1)} - \mathbf{c}^T \mathbf{v}^{(1)} \\ & - \mathbf{b}^T \mathbf{h}^{(1)} - (\mathbf{h}^{(2)})^T \mathbf{W} \mathbf{v}^{(2)} - \mathbf{c}^T \mathbf{v}^{(2)} - \mathbf{b}^T \mathbf{h}^{(2)} \end{aligned} \quad (3.8)$$

where  $(\mathbf{h}^{(1)})^T \mathbf{M} \mathbf{h}^{(2)}$  forces the latent feature representations of  $\langle u_1, u_2 \rangle$  to be close. Obviously, if  $\mathbf{M}$  is the identity matrix  $\mathbf{I}$ , then  $(\mathbf{h}^{(1)})^T \mathbf{M} \mathbf{h}^{(2)}$  reduces to  $(\mathbf{h}^{(1)})^T \mathbf{h}^{(2)}$ , which is the similarity between  $\mathbf{h}^{(1)}$  and  $\mathbf{h}^{(2)}$ . In this way, learning  $\mathbf{M}$  can help us to capture more complex similarity while  $\mathbf{M} = \mathbf{I}$  is a special case. The vectors  $\mathbf{v}^{(1)}$  and  $\mathbf{v}^{(2)}$  are the original feature vectors of a pair of instances, and the scalar  $a$  is the weight of the link between  $\mathbf{v}^{(1)}$  and  $\mathbf{v}^{(2)}$ . For a pair of linked users,  $a$  is set to 1, which forces the latent representation of these two users to be similar. For a pair of non-linked users,  $a$  is set to 0. Then pRBM reduces to RBM and theirs not interaction between



**Figure 3.4:** An Illustration of Paired Restricted Boltzmann Machine for Attributed Network Embedding

these two users. The partition function is given as

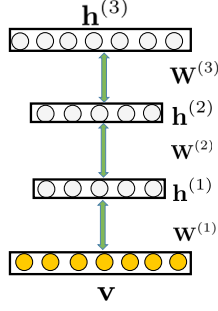
$$Z = \sum_{\mathbf{v}^{(1)}} \sum_{\mathbf{v}^{(2)}} \sum_{\mathbf{h}^{(1)}} \sum_{\mathbf{h}^{(2)}} \exp(-E(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)})) \quad (3.9)$$

And the marginal distribution  $P(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, a)$  is given as

$$P(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, a) = \sum_{\mathbf{h}^{(1)}} \sum_{\mathbf{h}^{(2)}} P(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, a; \boldsymbol{\theta}) \quad (3.10)$$

The paired Restricted Boltzmann Machine pRBM introduced in this work improves upon RBM for attributed networks and is different from Deep Boltzmann machines (DBM) [67]. An illustration of a 3 hidden-layer DBM is shown in Figure 3.5. From Figures 3.2, 3.4 and 3.5, we can see that the differences are

- RBM and 3 hidden-layer DBM work with independent data instances (or nodes); while pRBM works with pairs of linked data instances, which results in substantially different structures. RBM has one visible layer  $\mathbf{v}$  and one hidden layer  $\mathbf{h}$  as shown in Figure 3.2, the 3 hidden-layer DBM has one visible layer  $\mathbf{v}$  and three hidden layers  $\mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}$  for learning higher level features as shown in Figure 3.5, while pRBM has two visible layers  $\mathbf{v}^{(1)}, \mathbf{v}^{(2)}$  and two linked hidden layers  $\mathbf{h}^{(1)}, \mathbf{h}^{(2)}$  for modeling linked nodes as shown in Figure 3.4; and
- The 3 hidden-layer DBM stacks three RBMs and has different weights for each layer as shown in Figure 3.5; while pRBM shares weights for the two RBMs



**Figure 3.5:** An Illustration of 3 Hidden-layer DBM

and has weights to model the interaction between  $\mathbf{h}^{(1)}$  and  $\mathbf{h}^{(2)}$  as shown in Figure 3.4. Therefore, pRBM has fewer parameters to train and can model attributed networks.

### 3.3 Training pRBM

The training process of pRBM involves sampling from  $P(\mathbf{h}^{(1)}, \mathbf{h}^{(2)} | \mathbf{v}^{(1)}, \mathbf{v}^{(2)})$ . However, unlike RBM, because of the link between the two hidden layers  $\mathbf{h}^{(1)}$  and  $\mathbf{h}^{(2)}$  of pRBM, sampling from  $P(\mathbf{h}^{(1)}, \mathbf{h}^{(2)} | \mathbf{v}^{(1)}, \mathbf{v}^{(2)})$  becomes difficult, which makes training pRBM challenging. Next, we give details about how to train pRBM.

Given  $\mathbf{V}^{(1)}$ ,  $\mathbf{V}^{(2)}$ , and  $\mathbf{a}$ , the log-likelihood function of pRBM can be written as

$$l(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N \log P(\mathbf{v}_i^{(1)}, \mathbf{v}_i^{(2)}, a_i; \boldsymbol{\theta}) \quad (3.11)$$

We use the gradient ascent method to update the variables  $\mathbf{M}$ ,  $\mathbf{W}$ ,  $\mathbf{b}$  and  $\mathbf{c}$ . For simplicity of notation, let  $\mathbf{h} = \{\mathbf{h}^{(1)}, \mathbf{h}^{(2)}\}$  and  $\mathbf{v} = \{\mathbf{v}^{(1)}, \mathbf{v}^{(2)}\}$ . Then the marginal distribution  $P(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, a)$  is written as  $P(\mathbf{v}, a)$ . The derivative of  $\log P(\mathbf{v}, a)$  with respect to  $\mathbf{W}$  is:

$$\begin{aligned} \frac{\partial \log P(\mathbf{v}, a)}{\partial \mathbf{W}} &= \sum_{\mathbf{h}} P(\mathbf{h} | \mathbf{v}) [\mathbf{h}^{(1)} (\mathbf{v}^{(1)})^T + \mathbf{h}^{(2)} (\mathbf{v}^{(2)})^T] \\ &\quad - \sum_{\mathbf{h}} \sum_{\tilde{\mathbf{v}}} P(\mathbf{h}, \tilde{\mathbf{v}}) [\mathbf{h}^{(1)} (\tilde{\mathbf{v}}^{(1)})^T + \mathbf{h}^{(2)} (\tilde{\mathbf{v}}^{(2)})^T] \end{aligned} \quad (3.12)$$

Thus, the derivative of the objective function in Eq.(3.11) w.r.t  $\mathbf{W}$  can be written as

$$\frac{\partial l(\boldsymbol{\theta})}{\partial \mathbf{W}} = \mathbb{E}_{P_{data}} [\mathbf{h}^{(1)}(\mathbf{v}^{(1)})^T + \mathbf{h}^{(2)}(\mathbf{v}^{(2)})^T] - \mathbb{E}_{P_{model}} [\mathbf{h}^{(1)}(\mathbf{v}^{(1)})^T + \mathbf{h}^{(2)}(\mathbf{v}^{(2)})^T] \quad (3.13)$$

In Eq. (3.13),  $\mathbb{E}_{P_{data}}[\cdot]$  denotes an expectation with respect to the data distribution

$$P_{data}(\mathbf{h}, \mathbf{v}^{(1)}, \mathbf{v}^{(2)}) = P(\mathbf{h}|\mathbf{v}^{(1)}, \mathbf{v}^{(2)})P_{data}(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}) \quad (3.14)$$

where  $P_{data}(\mathbf{v}^{(1)}, \mathbf{v}^{(2)})$  represents the empirical distribution as

$$P_{data}(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}) = \frac{1}{N} \sum_i \delta(\mathbf{v}^{(1)}, \mathbf{v}_i^{(1)})\delta(\mathbf{v}^{(2)}, \mathbf{v}_i^{(2)}), \quad (3.15)$$

and  $\delta(x, y)$  is the delta function whose value is 1 if  $x = y$  and 0 otherwise. In Eq. (3.13),  $\mathbb{E}_{P_{model}}[\cdot]$  is an expectation with respect to the distribution defined by the model, i.e.,  $P(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)})$ . Using the same procedure, the derivative of the objective function w.r.t  $\mathbf{b}$  is given by

$$\frac{\partial l(\boldsymbol{\theta})}{\partial \mathbf{b}} = \mathbb{E}_{P_{data}} (\mathbf{h}^{(1)} + \mathbf{h}^{(2)}) - \mathbb{E}_{P_{model}} (\mathbf{h}^{(1)} + \mathbf{h}^{(2)}). \quad (3.16)$$

Similarly, we can get the derivative of the objective function w.r.t  $\mathbf{c}$  as

$$\frac{\partial l(\boldsymbol{\theta})}{\partial \mathbf{c}} = \mathbb{E}_{P_{data}} (\mathbf{v}^{(1)} + \mathbf{v}^{(2)}) - \mathbb{E}_{P_{model}} (\mathbf{v}^{(1)} + \mathbf{v}^{(2)}) \quad (3.17)$$

The derivative of  $\log P(\mathbf{v}, a)$  w.r.t  $\mathbf{W}$  can be written as

$$\frac{\partial \log P(\mathbf{v}, a)}{\partial \mathbf{M}} = \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v})[a\mathbf{h}^{(1)}(\mathbf{h}^{(2)})^T] - \sum_{\mathbf{h}} \sum_{\tilde{\mathbf{v}}} P(\mathbf{h}, \tilde{\mathbf{v}})[\tilde{a}\mathbf{h}^{(1)}(\mathbf{h}^{(2)})^T] \quad (3.18)$$

where  $\tilde{\mathbf{v}} = \{\tilde{\mathbf{v}}^{(1)}, \tilde{\mathbf{v}}^{(2)}\}$  with  $\tilde{\mathbf{v}}^{(1)} \in \{0, 1\}^m$  and  $\tilde{\mathbf{v}}^{(2)} \in \{0, 1\}^m$ . The scalar  $\tilde{a}$  is the weight of link between  $\tilde{\mathbf{v}}^{(1)}$  and  $\tilde{\mathbf{v}}^{(2)}$ . Thus, for each pair of  $\tilde{\mathbf{v}}^{(1)}$  and  $\tilde{\mathbf{v}}^{(2)}$ , we need to estimate the corresponding  $\tilde{a}$ , which is intractable. We use  $\bar{a} = \frac{1}{N} \sum_i a_i$  to approximate  $\tilde{a}$ . Note that this approximation has no effects on unweighed links since

we always have  $a = 1$  in the energy function. Therefore, we can get the derivative of the objective function with respect to  $\mathbf{M}$  as

$$\frac{\partial l(\boldsymbol{\theta})}{\partial \mathbf{M}} = \frac{1}{N} \sum_{i=1}^N \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}_i^{(1)}, \mathbf{v}_i^{(2)}) [a_i \mathbf{h}^{(1)} (\mathbf{h}^{(2)})^T] - \bar{a} \mathbb{E}_{P_{model}} [\mathbf{h}^{(1)} (\mathbf{h}^{(2)})^T] \quad (3.19)$$

As with RBMs, in Eqs.(3.13), (3.16), (3.17) and (3.19), the second terms are referred to as negative gradient and the exact calculation of  $\mathbb{E}_{P_{model}}[\cdot]$  is intractable. Following the common way to deal with the negative gradient [50, 36], we use Persistent Contrastive Divergence (PCD) [82] to approximate  $\mathbb{E}_{P_{model}}[\cdot]$ . Specifically, Contrastive Divergence is to get samples of  $\mathbb{E}_{P_{model}}[\cdot]$  by starting a Gibbs chain at a training instance and run it for few steps [34]. Instead of using a new Gibbs Chain for each parameter, Persistent Contrastive Divergence is to use one Gibbs chain for all the parameters. With the approximation, the gradient of  $\mathbf{W}$  takes the form

$$\Delta \mathbf{W} = \mathbb{E}_{P_{data}} [\mathbf{h}^{(1)} (\mathbf{v}^{(1)})^T + \mathbf{h}^{(2)} (\mathbf{v}^{(2)})^T] - \mathbb{E}_{P_T} [\mathbf{h}^{(1)} (\mathbf{v}^{(1)})^T + \mathbf{h}^{(2)} (\mathbf{v}^{(2)})^T] \quad (3.20)$$

where  $P_T$  represents a distribution defined by running the Gibbs chain for T full steps [36]. Similarly, the gradient of  $\mathbf{b}$ ,  $\mathbf{c}$  and  $\mathbf{M}$  are

$$\begin{aligned} \Delta \mathbf{b} &= \mathbb{E}_{P_{data}} (\mathbf{h}^{(1)} + \mathbf{h}^{(2)}) - \mathbb{E}_{P_T} (\mathbf{h}^{(1)} + \mathbf{h}^{(2)}) \\ \Delta \mathbf{c} &= \mathbb{E}_{P_{data}} (\mathbf{v}^{(1)} + \mathbf{v}^{(2)}) - \mathbb{E}_{P_T} (\mathbf{v}^{(1)} + \mathbf{v}^{(2)}) \\ \Delta \mathbf{M} &= \frac{1}{N} \sum_i \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}_i^{(1)}, \mathbf{v}_i^{(2)}) (a_i \mathbf{h}^{(1)} (\mathbf{h}^{(2)})^T) - \bar{a} \mathbb{E}_{P_{model}} (\mathbf{h}^{(1)} (\mathbf{h}^{(2)})^T) \end{aligned} \quad (3.21)$$

To run Gibbs sampling, we need an efficient Gibbs sampler that alternates between sampling the states of the hidden units independently given the states of the visible units, and vice versa. From Figure 3.4, we can see that  $\mathbf{v}^{(1)}$  and  $\mathbf{v}^{(2)}$  are conditionally independent on  $\mathbf{h}^{(1)}, \mathbf{h}^{(2)}$ . So when  $\mathbf{h}^{(1)}$  and  $\mathbf{h}^{(2)}$  are given, we have

$$P(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}|\mathbf{h}^{(1)}, \mathbf{h}^{(2)}) = P(\mathbf{v}^{(1)}|\mathbf{h}^{(1)})P(\mathbf{v}^{(2)}|\mathbf{h}^{(2)}) \quad (3.22)$$

where  $P(\mathbf{v}^{(1)}|\mathbf{h}^1)$  has the same form as RBM

$$P(\mathbf{v}^{(1)}|\mathbf{h}^{(1)}) = \prod_{i=1}^m P(v_i^{(1)}|\mathbf{h}^{(1)}) \quad (3.23)$$

with

$$P(v_i^{(1)} = 1|\mathbf{h}^{(1)}) = \sigma(c_i + (\mathbf{h}^{(1)})^T \mathbf{W}_{.i}) \quad (3.24)$$

Similarly, we have

$$P(\mathbf{v}^{(2)}|\mathbf{h}^{(2)}) = \prod_{i=1}^m P(v_i^{(2)}|\mathbf{h}^{(2)}) \quad (3.25)$$

with

$$P(v_i^{(2)} = 1|\mathbf{h}^{(2)}) = \sigma(c_i + (\mathbf{h}^{(2)})^T \mathbf{W}_{.i}) \quad (3.26)$$

Therefore, sampling of the visible layers given the hidden layers is very efficient. However, since  $\mathbf{h}^{(1)}$  and  $\mathbf{h}^{(2)}$  are not independent given  $\mathbf{v}^{(1)}$  and  $\mathbf{v}^{(2)}$  (see Figure 3.4), the sampling of  $P(\mathbf{h}|\mathbf{v})$  becomes intractable when the dimension of  $\mathbf{h}^{(1)}$  and  $\mathbf{h}^{(2)}$  are large. We use mean-field inference to deal with this problem. Consider any approximation distribution  $Q(\mathbf{h}|\mathbf{v}; \boldsymbol{\mu})$ , parameterized by a vector of parameters  $\boldsymbol{\mu}$ , for the posterior  $P(\mathbf{h}|\mathbf{v}; \boldsymbol{\theta})$ . Then the likelihood of the pRBM model has the following variational lower bound [59]

$$\log P(\mathbf{v}; \boldsymbol{\theta}) \geq \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{v}; \boldsymbol{\mu}) \log P(\mathbf{v}, \mathbf{h}; \boldsymbol{\theta}) + H(Q) \quad (3.27)$$

where  $H(\cdot)$  is the entropy function. The bound becomes tight if and only if  $Q(\mathbf{h}|\mathbf{v}; \boldsymbol{\mu}) = P(\mathbf{h}|\mathbf{v}; \boldsymbol{\theta})$

For simplicity and efficiency, we approximate the true posterior  $P(\mathbf{h}|\mathbf{v}; \boldsymbol{\theta})$  with a fully factorized approximating distribution over the two sets of hidden units, which correspond to the mean-field approximation

$$Q^{MF}(\mathbf{h}|\mathbf{v}; \boldsymbol{\mu}) = \prod_{i=1}^d q_i^{(1)}(h_i^{(1)}) \prod_{j=1}^d q_j^{(2)}(h_j^{(2)}) \quad (3.28)$$



where  $d$  is the dimension of the hidden layer, and  $\boldsymbol{\mu} = \{\boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}\}$  are the mean-field parameters with  $q_i^{(1)}(h_i^{(1)} = 1) = \mu_i^{(1)}$  and  $q_j^{(2)}(h_j^{(2)} = 1) = \mu_j^{(2)}$ . With mean-field approximation, to maximize the lower bound of Eq.(3.27), we only need to set  $q_i^{(1)}$  as [59]

$$\log q_i^{(1)}(h_i^{(1)}) = \mathbb{E}_{-q_i}[\log \tilde{p}(\mathbf{h})] + \text{const} \quad (3.29)$$

where  $\mathbb{E}_{-q_i}[\log \tilde{p}(\mathbf{h})]$  is defined as

$$\mathbb{E}_{-q_i}[\log \tilde{p}(\mathbf{h})] = \sum_{\mathbf{h}_{-i}^{(1)}} \sum_{\mathbf{h}^{(2)}} \prod_{m \neq i} q_m^{(1)}(h_m^{(1)}) \prod_{j=1}^d q_j^{(2)}(h_j^{(2)}) \log \tilde{p}(\mathbf{h}) \quad (3.30)$$

and  $\tilde{p}(\mathbf{h}) \propto \exp(-E(\mathbf{v}, \mathbf{h}; \boldsymbol{\theta}))$  with all the variables being constant except  $h_i^{(1)}$  and  $\mathbf{h}_{-i}^{(1)}$  is  $\mathbf{h}^{(1)}$  except  $h_i^{(1)}$ . Thus,  $\mathbb{E}_{-q_i}[\log \tilde{p}(\mathbf{h})]$  can be calculated as

$$\begin{aligned} \mathbb{E}_{-q_i}[\log \tilde{p}(\mathbf{h})] &= \sum_{\mathbf{h}_{-i}^{(1)}} \sum_{\mathbf{h}^{(2)}} \prod_{m \neq i} q_m^{(1)}(h_m^{(1)}) \prod_{j=1}^d q_j^{(2)}(h_j^{(2)}) [-E(\mathbf{v}, \mathbf{h}; \boldsymbol{\theta})] \\ &= h_i^{(1)} [a \sum_j M_{ij} \mu_j^{(2)} + \sum_j W_{ij} v_j^{(1)} + b_i] \end{aligned} \quad (3.31)$$

From the above equation, we can get

$$q_i^{(1)}(h_i^{(1)}) \propto \exp \left( h_i^{(1)} [a \sum_j M_{ij} \mu_j^{(2)} + \sum_j W_{ij} v_j^{(1)} + b_i] \right) \quad (3.32)$$

Then the mean of  $q_i^{(1)}(h_i^{(1)})$  is given as,

$$\begin{aligned} \mu_i^{(1)} &= \frac{q_i^{(1)}(h_i^{(1)} = 1)}{q_i^{(1)}(h_i^{(1)} = 1) + q_i^{(1)}(h_i^{(1)} = 0)} \\ &= \sigma \left( a \sum_j M_{ij} \mu_j^{(2)} + \sum_j W_{ij} v_j^{(1)} + b_i \right) \end{aligned} \quad (3.33)$$

which can be written as

$$\boldsymbol{\mu}^{(1)} = \sigma(a\mathbf{M}\boldsymbol{\mu}^{(2)} + \mathbf{W}\mathbf{v}^{(1)} + \mathbf{b}). \quad (3.34)$$

With the same procedure, we can get that

$$q_j^{(2)}(h_j^{(2)}) \propto \exp \left( h_j^{(2)} [a \sum_j M_{ij} \mu_i^{(1)} + \sum_i W_{ji} v_i^{(2)} + b_j] \right) \quad (3.35)$$

and thus  $\boldsymbol{\mu}^{(2)}$  is estimated as

$$\boldsymbol{\mu}^{(2)} = \sigma(a(\boldsymbol{\mu}^{(1)})^T \mathbf{M} + \mathbf{W}\mathbf{v}^{(2)} + \mathbf{b}) \quad (3.36)$$

As we can see from Eq.(3.34) and Eq.(3.36), the update rules of  $\boldsymbol{\mu}^{(1)}$  and  $\boldsymbol{\mu}^{(2)}$  are fixed-point equations and are coupled together. To solve these fixed-point equations, we simply cycle through layers by updating the mean-field parameters within a single layer, *i.e.*, updating  $\boldsymbol{\mu}^{(1)}$  and  $\boldsymbol{\mu}^{(2)}$  alternatively by fixing one and update the other one until they converge. To make it smoother, we use damped updates as

$$\boldsymbol{\mu}^{(1)} \leftarrow \lambda \boldsymbol{\mu}^{(1)} + (1 - \lambda)(a\mathbf{M}\boldsymbol{\mu}^{(2)} + \mathbf{W}\mathbf{v}^{(1)} + \mathbf{b}) \quad (3.37)$$

$$\boldsymbol{\mu}^{(2)} \leftarrow \lambda \boldsymbol{\mu}^{(2)} + (1 - \lambda)(a(\boldsymbol{\mu}^{(1)})^T \mathbf{M} + \mathbf{W}\mathbf{v}^{(2)} + \mathbf{b}) \quad (3.38)$$

Given  $\boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}$ , we can sample  $\mathbf{h}^{(1)}, \mathbf{h}^{(2)}$  from the distribution  $Q^{MF}(\mathbf{h}|\mathbf{v}; \boldsymbol{\mu})$  efficiently. Then  $\mathbf{h}^{(1)}, \mathbf{h}^{(2)}$  are used to sample  $\mathbf{v}^{(1)}, \mathbf{v}^{(2)}$  using Eq.(3.24) and Eq.(3.26), which gives us a Gibbs chain. Thus, with mean-field inference, we can perform Gibbs sampling efficiently, which allows us to calculate both  $\mathbb{E}_{P_{data}}[\cdot]$  and  $\mathbb{E}_{P_T}[\cdot]^2$ . The training algorithm for pRBM is summarized in Algorithm 1. Next we briefly review Algorithm 1. We first pretrain pRBM by using RBM to initialize  $\mathbf{W}, \mathbf{b}, \mathbf{c}$ . After that, we prepare the paired training data  $\mathbf{V}^{(1)}, \mathbf{V}^{(2)}$  and  $\mathbf{a}$  from the attribute-value matrix  $\mathbf{V}$  and the adjacency matrix  $\mathbf{A}$ . Since we use mini-batch training, we split the data into mini-batches. For each mini-batch, we perform PCD by running Gibbs sampling with mean-field inference to calculate the gradients, *i.e.*,  $\Delta \mathbf{W}, \Delta \mathbf{b}, \Delta \mathbf{c}$  and  $\Delta \mathbf{M}$ . With the gradients, we update  $\mathbf{W}, \mathbf{b}, \mathbf{c}$  and  $\mathbf{M}$  using gradient descent where  $\epsilon$  is the learning rate.

---

<sup>2</sup>The calculation of  $\mathbb{E}_{P_{data}}[\cdot]$  also depends on the calculation of  $P(\mathbf{h}|\mathbf{v})$ . For simplicity and efficiency, we also use mean-field inference to approximate

---

**Algorithm 1** Training pRBM

---

**Require:**  $\mathbf{V} \in \mathbb{R}^{m \times n}$ ,  $\mathbf{A} \in \mathbb{R}^{n \times n}$ ,  $d$

**Ensure:**  $\mathbf{M} \in \mathbb{R}^{d \times d}$ ,  $\mathbf{W} \in \mathbb{R}^{d \times m}$ ,  $\mathbf{b} \in \mathbb{R}^d$ ,  $\mathbf{c} \in \mathbb{R}^m$

- 1: initialize  $\mathbf{W}$ ,  $\mathbf{b}$ ,  $\mathbf{c}$  using an RBM
  - 2: prepare paired data  $\mathbf{V}^{(1)}$ ,  $\mathbf{V}^{(2)}$ ,  $\mathbf{a}$  from  $\mathbf{V}$  and  $\mathbf{A}$
  - 3: **for** epoch = 1:maxepoch **do**
  - 4:     **for** batch = 1:numbatch **do**
  - 5:         perform PCD by running Gibbs sampling with mean-field inference
  - 6:         calculate  $\Delta\mathbf{W}$  using Eq.(3.20)
  - 7:         calculate  $\Delta\mathbf{b}$ ,  $\Delta\mathbf{c}$ , and  $\Delta\mathbf{M}$  using Eq.(3.21)
  - 8:         update  $\mathbf{W}$  as  $\mathbf{W} = \mathbf{W} + \epsilon\Delta\mathbf{W}$ ,
  - 9:         update  $\mathbf{b}$  as  $\mathbf{b} = \mathbf{b} + \epsilon\Delta\mathbf{b}$
  - 10:         update  $\mathbf{c}$  as  $\mathbf{c} = \mathbf{c} + \epsilon\Delta\mathbf{c}$ ,
  - 11:         update  $\mathbf{M}$  as  $\mathbf{M} = \mathbf{M} + \epsilon\Delta\mathbf{M}$
  - 12:     **end for**
  - 13: **end for**
- 

### 3.3.1 Representation Learning with pRBM

After pRBM is trained, learning representations is equivalent to sampling from the posterior distribution  $P(\mathbf{h}^{(1)}, \mathbf{h}^{(2)} | \mathbf{v}^{(1)}, \mathbf{v}^{(2)})$ . As we discussed in the previous subsection, because of the dependence between  $\mathbf{h}^{(1)}$  and  $\mathbf{h}^{(2)}$ , the sampling of  $P(\mathbf{h}^{(1)}, \mathbf{h}^{(2)} | \mathbf{v}^{(1)}, \mathbf{h}^{(2)})$  is very difficult. Thus, we use the same method, *i.e.*, mean-field approximation, to infer this posterior distribution. Specifically, if we want to get the feature representation of  $u_i$  whose corresponding input feature vector is  $\mathbf{v}_i$ , we first find the set  $\mathcal{U}_i = \{u_j : u_i \text{ is connected to } u_j\}$ . For each  $u_j \in \mathcal{U}_i$ , we can get the input data  $(\mathbf{v}_i, \mathbf{v}_j, A_{ij})$ . Ideally, we want to sample from  $P(\mathbf{h}_{ij}, \mathbf{h}_j | \mathbf{v}_i, \mathbf{v}_j)$  to get  $\mathbf{h}_{ij}$  where  $\mathbf{h}_{ij}$

means that the feature representation is from  $(\mathbf{v}_i, \mathbf{v}_j, A_{ij})$ . Due to the reason that sampling from  $P(\mathbf{h}_{ij}, \mathbf{h}_j | \mathbf{v}_i, \mathbf{v}_j)$  is difficult, we use the mean-field inference instead. We first calculate  $\boldsymbol{\mu}^{(1)}$  and  $\boldsymbol{\mu}^{(2)}$  using Eq.(3.37) and Eq.(3.38) with  $\mathbf{v}_i, \mathbf{v}_j$  as input. We then sample  $\mathbf{h}_{ij}$  from  $Q^{MF}(\mathbf{h}_{ij}, \mathbf{h}_j | \mathbf{v}_i, \mathbf{v}_j; \boldsymbol{\mu})$ , whose definition is given by Eq.(3.28). Finally, the feature representation of  $u_i$  is given by the weighted sum as

$$\bar{\mathbf{h}}_i = \frac{\sum_{j:u_j \in \mathcal{U}_i} A_{ij} \mathbf{h}_{ij}}{\sum_{j:u_j \in \mathcal{U}_j} A_{ij}} \quad (3.39)$$

### 3.3.2 Time Complexity

Given a training instance  $\mathbf{v}^{(1)}, \mathbf{v}^{(2)}$ , we need to perform PCD, which involves running Gibbs sampling using mean-field approximation. The main cost of Gibbs sampling is on the calculation of  $\boldsymbol{\mu}^{(1)}$  and  $\boldsymbol{\mu}^{(2)}$ , which is  $\mathcal{O}(d^2 + dm)$ . Thus, it takes  $\mathcal{O}(d^2 + dm)$  to perform Gibbs sampling with mean-field inference for one pair of training instance. The total computational cost of  $\Delta \mathbf{W}, \Delta \mathbf{b}, \Delta \mathbf{c}$ , and  $\Delta \mathbf{M}$  using Eq.(3.21) and Eq.(3.20) is also  $\mathcal{O}(d^2 + dm)$ . Similarly, the total time complexity of updating  $\mathbf{W}, \mathbf{b}, \mathbf{c}, \mathbf{M}$  using gradient ascent as shown in line 8 to line 11 in Algorithm 1 is  $\mathcal{O}(d^2 + dm)$ . Since there are  $N$  pairs of training instances, the computational cost of each epoch is  $\mathcal{O}(Nd^2 + Ndm)$ .

## 3.4 Evaluating pRBM

In this section, we conduct experiments to evaluate the effectiveness of pRBM for attributed network embedding. Specifically, we aim to answer the following questions:

- Is the proposed framework pRBM effective in learning useful representations for attributed networks?
- How robust is pRBM when datasets are small?

To answer these questions, we conduct extensive experiments on two real-world datasets and compare the proposed framework pRBM with state-of-the-art algorithms. We begin by explaining the experimental setting.

### 3.4.1 Experimental Settings

We use two datasets from real-world social media websites, *i.e.*, BlogCatalog<sup>3</sup> and Flickr<sup>4</sup>. These datasets are publicly available datasets used in [76] to study unsupervised feature selection for attributed networks. The statistics of the datasets are shown in Table 3.1. In both datasets, the number of links is much larger than that of data instances, thus, links have potential to provide extra information over attributes; and the number of features is larger than that of data instances, thus, it is necessary to learn dense representations. These characteristics make these two datasets suitable to assess the performance of unsupervised representation learning methods for attributed networks.

	# nodes	# links	Avg Degree	# Features	# Classes
BlogCatalog	5,198	27,965	5.38	8,189	6
Flickr	7,575	47,344	6.25	12,407	9

**Table 3.1:** Statistics of the Attributed Social Network Datasets.

Following the common way to assess the performance of unsupervised representation learning algorithms, we use clustering performance to evaluate the quality of learned representations. Intuitively, better representations will lead to better clustering performance. Each unsupervised representation learning algorithm is first performed to learn feature representations, and then  $k$ -means clustering is performed

<sup>3</sup><http://www.blogcatalog.com>

<sup>4</sup><http://www.flickr.com>

based on the learned features. The clustering quality is evaluated by two commonly used metrics: *accuracy* (ACC) and *normalized mutual information* (NMI).

### 3.4.2 Quality of Learned Representations

In order to answer the question of “is the proposed framework pRBM effective in learning useful representations by exploiting node attributes and network information?”, we assess the quality of representations learned by different representation learning algorithms via clustering performance. pRBM is compared with the following representative and state-of-the-art methods:

- ALL: We perform clustering on the original data without representation learning.
- PCA: Principle Component Analysis [39] performs dimensionality reduction by seeking orthogonal projections of the data onto a low-dimensional linear space such that the variance of the projected data is maximized. It is a popular and effective linear feature learning algorithm. We use it as a representative traditional representation learning algorithm.
- DAE: Denoising autoencoder [87] is a variant of autoencoder that is to learn a feature representation that is able to reconstruct the input data. Specifically, DAE is trained to reconstruct a clean “repaired” input from a corrupted version, which makes it able to extract more robust features. The encoded feature is used to perform clustering. We use it as a representative nonlinear feature learning algorithm.
- SDAE: Stacked denoising autoencoder [88] is a deep network based on stacking layers of denoising autoencoders which are trained locally to denoise corrupted

versions of their inputs. Compared with the denoising autoencoder, features learned in a purely unsupervised fashion by SDAE are higher-level and could boost the performance of clustering. We used a three-layer stacked denoising autoencoder and the third layer feature representation is used for clustering in our experiment. SDAE is used as a representative deep learning algorithm for unsupervised representation learning.

- RBM: Restricted Boltzmann machine [22] is an undirected graphical model which defines a probability distribution over a vector of observed and a vector of latent variables. The learned latent variable is used for clustering in our experiment. RBM can be seen as pRBM without link information.
- RTM: Relational Topic Model [13] is a variant of Latent Dirichlet Allocation (LDA), which takes attribute and link information into consideration for learning topic distributions. The learned topic distributions of documents are treated as the representations.
- TADW: Text-associated DeepWalk [96] incorporates both attribute and link information into the matrix factorization framework to learn representations of each nodes. It is state-of-the-art representation learning algorithm for network with rich attributes.
- LRBM: LRBM [50] combines graph factorization and conditional RBM using four-way tensor for social networks. It is the closest work to ours and their differences will be detailed in the related work section.

We use the “grid” search method to determine the values of parameters of the unsupervised representation learning algorithms. For the proposed model, we empirically set the number of hidden units to be 500 for both datasets. For each method,

Method	ALL	PCA	DAE	RBM	SDAE	RTM	TADW	LRBM	pRBM
BlogCatalog	36.00	42.45	53.73	53.60	55.77	54.35	54.76	49.60	<b>56.79</b>
Flickr	53.82	58.00	54.81	59.71	59.74	55.38	58.87	56.71	<b>61.95</b>

**Table 3.2:** Accuracy(%) Comparison on BlogCatalog and Flickr.

Method	ALL	PCA	DAE	RBM	SDAE	RTM	TADW	LRBM	pRBM
BlogCatalog	0.2176	0.2787	0.4047	0.3829	0.4078	0.3802	0.3954	0.3547	<b>0.4142</b>
Flickr	0.4334	0.4601	0.4459	0.4646	0.4831	0.4472	0.4553	0.4481	<b>0.5659</b>

**Table 3.3:** NMI Comparison on BlogCatalog and Flickr

we first learn feature representations and then use  $k$ -means clustering. Since the results of  $k$ -means depend on the initialization, we repeat each experiment 20 times and report the average performance. The comparison results, *i.e.*, accuracy and NMI performance in Flickr and BlogCatalog, are shown in Table 3.2 and 3.3. From the tables, we make the following observations:

- The performance of representation learning methods outperforms ALL, *i.e.*, using all features for clustering without learning representations, which suggests that representation learning can improve the performance.
- pRBM obtains better performance than RBM in both datasets. For example, on BlogCatalog dataset, pRBM gains 5.95% relative accuracy improvement and 8.17% relative NMI improvement compared to RBM. The performance improvement of pRBM compared with RBM demonstrates that link information does provide complementary information that could help learn better representations.
- SDAE, RBM and DAE outperform PCA, which suggests nonlinear features



learned by SDAE, RBM and DAE are more effective than linear features learned by PCA. In addition, SDAE outperforms DAE and RBM. SDAE is a deep network by stacking DAEs, which is able to learn more effective high-level representations. However, pRBM outperforms SDAE, which is because pRBM leverages both attribute and link information while SDAE only learns representations from attribute information.

- Though RTM, TADW and LRBM consider both attribute and link information, pRBM obtains better performance than them. We perform t-test on these results, which suggests that the improvement is significant. These results suggest that pRBM is more effective in leveraging both information for learning representations. In particular, LRBM, which utilizes conditional RBM, doesn't perform as well as RBM. RBM shares parameter for each data instance, i.e., the parameter have dimension  $\mathbf{W} \in \mathbf{R}^{d \times m}$ , where  $d$  is the number of latent dimensions and  $m$  is the number of attributes, thus we can still learn good representations from data with high dimensionality, i.e.,  $m \leq n$ , where  $n$  is the data size. However, for LRBM, the parameter is a four-way tensor that is much more complex. Therefore, given the small and sparse characteristics of the used datasets, LRBM doesn't perform well.

From these findings, we can draw a positive answer to the first question - pRBM is effective in learning representations by exploiting both attribute information and link information.

### 3.4.3 Robustness of pRBM to Small Data

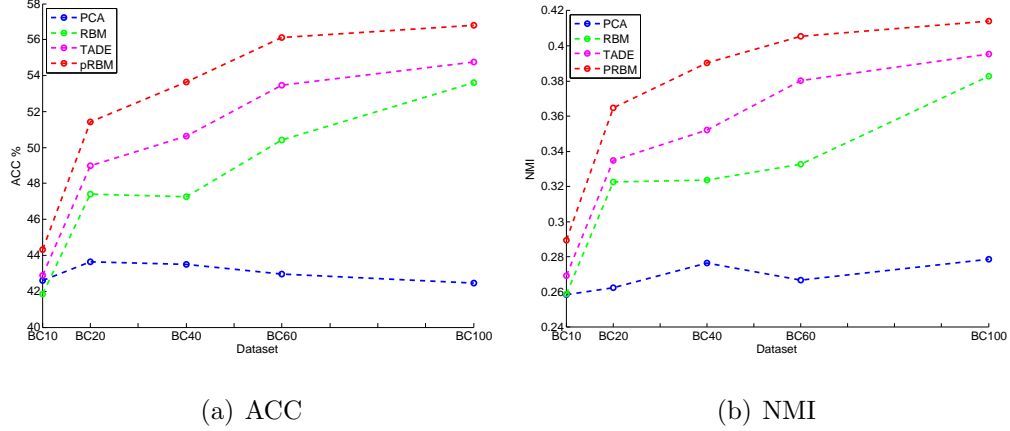
To answer the question of “how robust is pRBM when datasets are small?”, we examine how the performance of pRBM varies with changes to the size of data. To

achieve this goal, we randomly select  $x\%$  of the data instances from each class to construct smaller datasets from original datasets. We vary  $x$  as  $\{10, 20, 40, 60\}$  in the paper and correspondingly we can get four smaller datasets from each original dataset. For example, we construct BC10, BC20, BC40 and BC60 from BlogCatalog by randomly selecting 10%, 20%, 40% and 60% of its data instances. Furthermore, we compare each reduced dataset with the full dataset, named BC100. Since we make similar observations on both BlogCatalog and Flickr, we only report results on BlogCatalog. The statistics of these four datasets are shown in Table 3.4, where “Ratio” in the table refers to feature dimension over data size. Generally, it can also be used as a measure of how large the dataset is. A large ratio of feature dimensions as a function of the data size usually implies a small dataset.

	BC10	BC20	BC40	BC60
Size	523	1042	2081	3121
Features	8189			
Classes	6			
Links	2,782	5,480	10,717	16,885
Avg Degree	5.32	5.26	5.15	5.41
Ratio	15.66	7.86	3.94	2.62

**Table 3.4:** Statistics of the Reduced Datasets

From the table, we can see that though the number of instances is small, we still have a relatively large number of links, which could be sufficient to train pRBM. Similarly, we use accuracy and NMI clustering performance to assess the quality of learned features and  $k$ -means is chosen as the basic clustering algorithm. Since the results of  $k$ -means depend on the initialization, we repeat each experiment 20 times and report the average performance. The performance variances w.r.t. the size of data



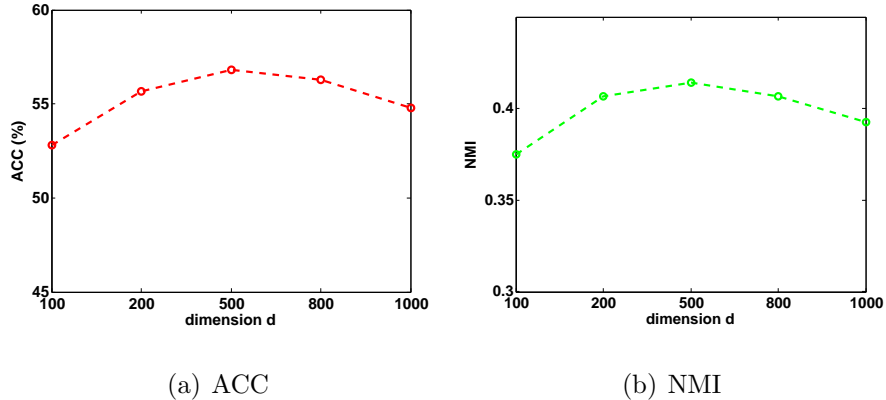
**Figure 3.6:** The Impact of the Size of Data on the Performance of Representation Learning Methods.

are shown in Figures 3.6(a) and 3.6(b) for accuracy and NMI, respectively. We also show the results of PCA, RBM and TADW for comparison because PCA represents linear representation learning algorithm, RBM can be seen as pRBM without link information and TADW is the state-of-the-art method for attributed networks. From the figures, it can be observed:

- In general, traditional feature learning algorithm PCA is stable with the changes of the size of data; while the performances of RBM, TADW and pRBM increase with the increase of the data size. This suggests that with larger dataset, RBM, TADW and pRBM can be better trained.
- When data size is small such as BC10, we cannot observe performance improvement from RBM compared to PCA; while pRBM and TADW always outperform PCA, which supports that link information is useful for representation learning when data size is small.
- In addition, pRBM always outperforms TADW. We also perform t-test on these results, which suggests that the improvement is significant. This implies that pRBM is more effective in leveraging link data and pRBM is also robust to

small dataset.

### 3.4.4 Parameter Sensitivity



**Figure 3.7:** Sensitivity of pRBM to Dimensionality

In this subsection, we investigate the impact of the dimensionality of the latent representations,  $d$ , on the performance of pRBM. We test  $d$  at  $\{100, 200, 500, 800, 1000\}$ . For each dataset, we first apply pRBM to learn representations and then perform clustering to assess the quality of the representations. We repeat each experiment 20 times and report the average performance. Since we make similar observations on both BlogCatalog and Flickr, we report results on BlogCatalog. The performance variances w.r.t  $d$  are shown in Figures 3.7(a) and 3.7(b). From the figures, we can see that, generally, as the dimensionality of the latent representation increases, the performance first increases until it reaches a certain point, then the performance decreases. For the two datasets used, we find that a value of  $d$  between 300 to 700 works well, which eases parameter selection.

### SIGNED NETWORK EMBEDDING

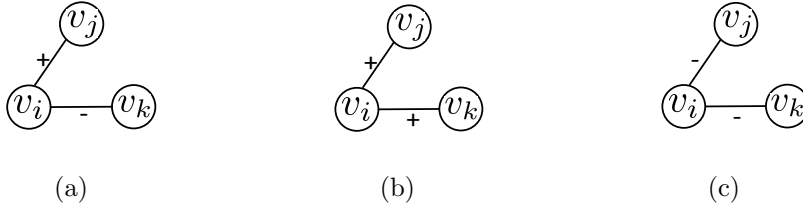
In this chapter, we study signed social network embedding. Social networks can contain both positive and negative links, and these signed social networks are present on a variety of social media sites, such as Epinions with trust and distrust links, and Slashdot with friend and foe links. In addition to existing signed social networks, many algorithms are proposed to construct signed networks from positive and negative interactions between users or documents [53, 31]. The availability of negative links in signed networks challenges some principles that explain the formation and properties of links for unsigned social networks; and principles for signed social networks can be substantially different from that of unsigned network [49, 73]. For example, homophily effects and social influence for unsigned networks may not be applicable to signed networks [74]. Therefore, signed network embedding cannot be carried out by simply extending embedding algorithms for unsigned social networks. Recent research on mining signed social networks suggests that negative links have added value over positive links in various analytical tasks. For example, a small number of negative links can significantly improve positive link prediction performance [48], and they can also improve recommendation performance in social media [72]. While signed network embedding is challenging, the results of such an approach have the potential to greatly advance tasks of mining signed social networks such as link prediction. Next, we propose a novel signed network embedding algorithm.

## 4.1 An Objective Function for Signed Network Embedding

Recent research about signed social networks suggests that negative links present distinct properties from positive links, and the fundamental principles that drive the formation of links for signed and unsigned social networks are very different [74, 49]. This suggests that we need a new objective function for signed network embedding because we cannot apply those for unsigned social networks directly on signed social networks.

Social theories are developed by social scientists to explain social phenomenon in signed social networks and they provide fundamental understandings about signed social networks. Social theories have been widely exploited in various tasks of mining signed social networks such as link prediction [49] and community detection [14]. The successful experiences on exploiting social theories in mining signed social networks suggest that social theories may guide us to develop objective functions for signed network embedding. Actually, social theories for unsigned social networks have been widely used to design objective functions for unsigned social network embedding. For example, social correlation theories such as homophily [95] and social influence [83] suggest that two connected users are likely to share similar interests, which are the foundations of many objective functions of unsigned network embedding [79, 28]. Inspired by the success of applying social theories in unsigned network, we seek social theories on signed network for signed network embedding. Among the social theories, structural balance theory is one of the most important and popular theories for signed social networks. Thus, in this work we develop an objective function for signed network embedding based on it.

Structural balance theory was originally proposed in [33] at the individual level, generalized by Cartwright and Harary [11] in the graph-theoretical formation at the



**Figure 4.1:** Three Types of Triplets of Users.

group level and then was developed to the concept of clusterizable graph in [16]. It is recently extended by [15] as: a structure in signed social network should ensure that users should be able to have their “friends” closer than their “foes”, i.e., users should sit closer to their “friends” (or users with positive links) than their “foes” (or users with negative links). In other words, the key idea of extended structural balance theory suggests that a user should be more similar to her friends than her foes. The extended structural balance theory provides us a guidance to model signed social network for learning network embedding. We will now introduce the detail of how we model signed social network based on extended structural balance theory.

Let  $\mathcal{P}$  be a set of triplets  $(v_i, v_j, v_k)$  as shown in Figure 4.1(a) from a given signed social network  $\mathcal{G}$ , where  $v_i$  and  $v_j$  have a positive link while  $v_i$  and  $v_k$  have a negative link. Formally,  $\mathcal{P}$  is defined as:

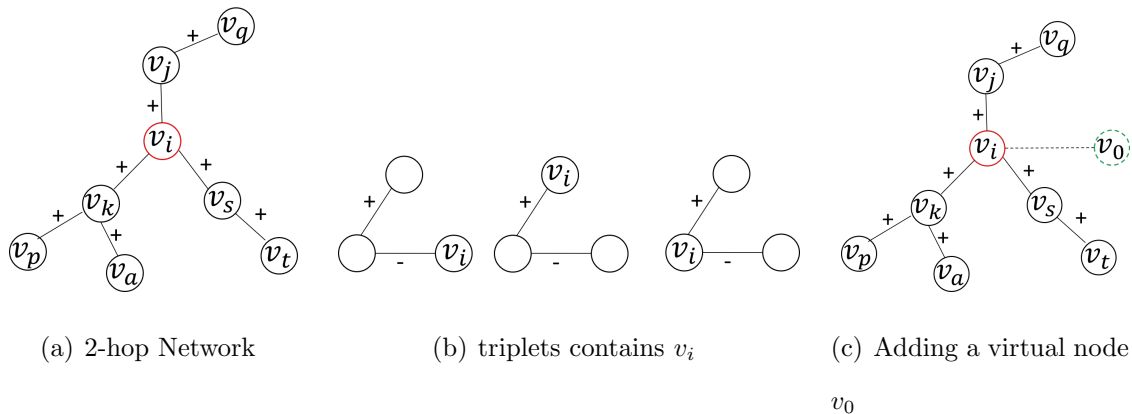
$$\mathcal{P} = \{(v_i, v_j, v_k) | e_{ij} = 1, e_{ik} = -1, v_i, v_j, v_k \in \mathcal{V}\},$$

The extended structural balance theory in [15] suggests that with a certain similarity measurement, for a triplet  $(v_i, v_j, v_k) \in \mathcal{P}$ ,  $v_i$  is likely to be more similar to the user with a positive link, i.e.  $v_j$ , than a user with a negative link, i.e.  $v_k$ , which can be mathematically modeled as:

$$f(\mathbf{x}_i, \mathbf{x}_j) \geq f(\mathbf{x}_i, \mathbf{x}_k) + \delta, \quad (4.1)$$

where  $\mathbf{x}_i$ ,  $\mathbf{x}_j$  and  $\mathbf{x}_k$  are the  $d$ -dimensional vector representations of  $v_i$ ,  $v_j$  and  $v_k$  respectively, which we need to learn by the proposed embedding framework. In

$f(\mathbf{x}_i, \mathbf{x}_j)$ ,  $f$  is a function that measures the similarity between  $\mathbf{x}_i$  and  $\mathbf{x}_j$ . We will discuss more details about the function  $f$  in the proposed framework in the following subsection. The parameter  $\delta$  is a threshold that is used to regulate the difference between these two similarities. A large  $\delta$  will push  $v_i, v_j$  more close and  $v_i, v_k$  more far away. The range of  $\delta$  will be discussed in experimental analysis section.

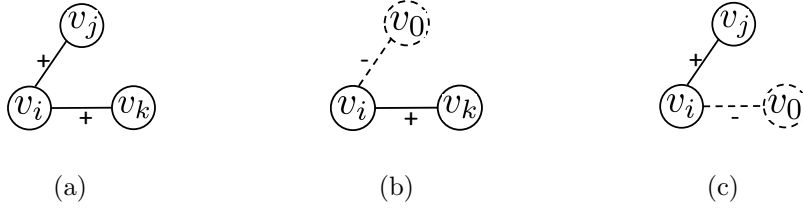


**Figure 4.2:** Dealing with Special Case

In a real-world signed network, the objective function in Eq. (4.1) has no effect on those nodes whose 2-hop networks<sup>1</sup> have only positive or negative links. Figure 4.2(a) gives an example, where  $v_i$ 's 2-hop network only has positive links. Then  $v_i$  will not be included in any triplets that has one negative link with  $v_i$  as shown in Figure 4.2(b). That is to say, we cannot learn the  $d$ -dimensional vector representations for those nodes whose 2-hop networks are all positive or negative because there are no triplets in  $\mathcal{P}$  that contains them. Those nodes are involved in triplets as shown in Figures 4.1(b) and Figures 4.1(c). According to a recent study [74], the cost of forming negative links is higher than that of forming positive links in social media. Therefore, in a signed social network, positive links are denser than negative links. This determines that there are many nodes whose 2-hop networks have only positive links while very

<sup>1</sup> $v_i$ 's 2-hop network is defined as the network formed by  $v_i$ , users whose distance from  $v_i$  is within 2 hops and links among them.





**Figure 4.3:** Adding a Virtual Node.

few nodes whose 2-hop networks have only negative links. Therefore, next we only consider handling nodes whose 2-hop networks have only positive links although a similar solution can be applied to dealing with the other type of nodes.

We first introduce a virtual node  $v_0$  and then create a negative link between  $v_0$  and each node whose 2-hop network has only positive links. For example, after adding a virtual node to  $v_i$  as shown in Figure 4.2(c), we can extract triplets such as  $(v_i, v_j, v_0)$  and  $(v_i, v_k, v_0)$  as shown in Figures 4.3(b) and 4.3(c). Let  $\mathcal{P}_0$  be the set of triplets  $(v_i, v_j, v_0)$  where  $v_i$  and  $v_j$  have a positive link while  $v_i$  and  $v_0$  have a negative link, and a similar objective function as Eq. (4.1) can be developed as:

$$f(\mathbf{x}_i, \mathbf{x}_j) \geq f(\mathbf{x}_i, \mathbf{x}_0) + \delta_0, \quad (4.2)$$

where  $\delta_0$  is a threshold to regulate the similarities. The reason of using  $\delta_0$  in Eq.(4.2) and  $\delta$  in Eq.(4.1) is that we can have more flexibility to distinguish triplets with or without the virtual node by tuning  $\delta$  and  $\delta_0$ . By adding the virtual node, we can make a node  $v_i$  whose 2-hop network contains only positive links closer to their neighbors.

## 4.2 The Proposed Framework SiNE

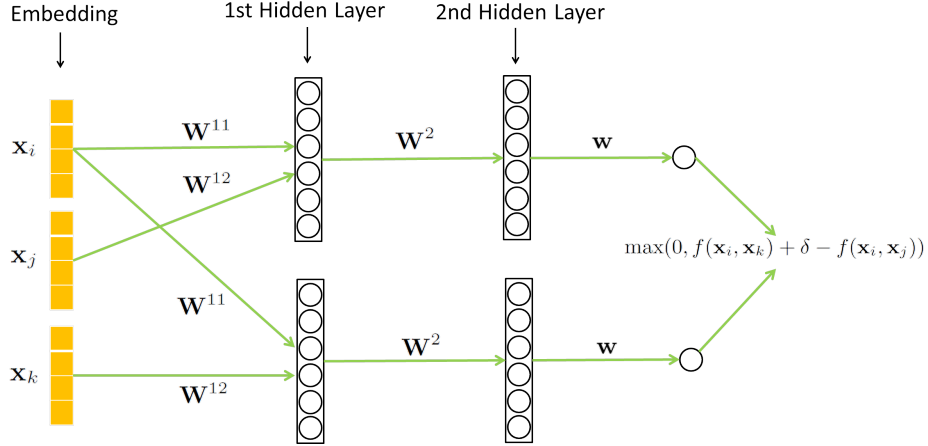
Based on Eq.(4.1) and (4.2), the objective function for signed social network embedding guided by the extended structural balance theory can be written as:

$$\begin{aligned} \min_{\mathbf{X}, \mathbf{x}_0, \theta} \quad & \frac{1}{C} \left[ \sum_{(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \in \mathcal{P}} \max(0, f(\mathbf{x}_i, \mathbf{x}_k) + \delta - f(\mathbf{x}_i, \mathbf{x}_j)) \right. \\ & \left. + \sum_{(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_0) \in \mathcal{P}_0} \max(0, f(\mathbf{x}_i, \mathbf{x}_0) + \delta_0 - f(\mathbf{x}_i, \mathbf{x}_j)) \right] + \alpha (\mathfrak{R}(\theta) + \|\mathbf{X}\|_F^2 + \|\mathbf{x}_0\|_2^2), \end{aligned} \quad (4.3)$$

where  $C = |\mathcal{P}| + |\mathcal{P}_0|$  is the size of the training data and  $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$  is the low-dimensional representation of the  $m$  nodes, and  $\theta$  is a set of parameters to define the similarity function  $f$ .  $\mathfrak{R}(\theta)$  is the regularizer to avoid overfitting and  $\alpha$  is a parameter to control the contribution of the regularizers.

## 4.3 The Architecture of SiNE

With the objective function given above, the task now is to find a function  $f$  that is able to give good similarity measure and learn good representations of nodes in signed network. Since signed networks are highly nonlinear, one choice of  $f$  is nonlinear functions, which have shown to be superior than linear functions for similarity measure and representation learning [6]. Among various non-linear functions, deep learning has been proven to be the state-of-the-art and very powerful for nonlinear representation learning [70, 6]. This suggests us to utilize the power of deep learning for learning nonlinear embedding of the nodes. In particular, we design a deep learning framework SiNE, which defines  $f$  with  $\theta$  and optimizes the objective function in Eq. (4.3). To help better understand SiNE, we first work on an illustrative example of the architecture of the proposed deep learning framework with 2 hidden layers (see Figure 4.4) and then generalize it to  $N$  layers. Note that we do not show bias in the



**Figure 4.4:** An Illustration of the Architecture of SiNE with 2 Hidden Layers.

figure. The input to the framework is the set of triplets extracted from the signed social network as  $(v_i, v_j, v_k)$  with  $e_{ij} = 1$  and  $e_{ik} = -1$ . The model is composed of two deep networks that share the same parameters. The outputs of the first hidden layer of the two deep networks (or “1st Hidden Layer ” in Figure 4.4) are given as:

$$\mathbf{z}^{11} = \tanh(\mathbf{W}^{11}\mathbf{x}_i + \mathbf{W}^{12}\mathbf{x}_j + \mathbf{b}^1), \quad \mathbf{z}^{12} = \tanh(\mathbf{W}^{11}\mathbf{x}_i + \mathbf{W}^{12}\mathbf{x}_k + \mathbf{b}^1) \quad (4.4)$$

where  $\tanh$  is the hyperbolic tangent function, which is one of the most widely used activation function in deep networks.  $\mathbf{W}^{11}$  and  $\mathbf{W}^{12}$  are the weights of the first hidden layer and  $\mathbf{b}^1$  is the bias.  $\mathbf{z}^{11}$  and  $\mathbf{z}^{12}$  are then used as inputs to the second hidden layer (or “2nd Hidden Layer” in Figure 4.4) of the two deep networks, separately. Similarly, the outputs of the second layer are  $\mathbf{z}^{21} = \tanh(\mathbf{W}^2\mathbf{z}^{11} + \mathbf{b}^2)$  and  $\mathbf{z}^{22} = \tanh(\mathbf{W}^2\mathbf{z}^{12} + \mathbf{b}^2)$ .  $f(\mathbf{x}_i, \mathbf{x}_j)$  and  $f(\mathbf{x}_i, \mathbf{x}_k)$  are the output of the two deep networks:

$$f(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\mathbf{w}^T\mathbf{z}^{21} + b), \quad f(\mathbf{x}_i, \mathbf{x}_k) = \tanh(\mathbf{w}^T\mathbf{z}^{22} + b) \quad (4.5)$$

which are the terms in Eq. (4.3) and the vector  $\mathbf{w}$  is the weights and the scalar  $b$  is the bias<sup>2</sup>. With the illustration of the proposed framework with 2 hidden layers,

<sup>2</sup>Note that for the proposed framework SiNE, the weights and bias to generate output  $f(\mathbf{x}_i, \mathbf{x}_j)$  are a vector and a scalar, separately

we can see that the similarity function  $f$  is defined by the deep network with a set of parameters as shown in Figure 4.4. Particularly, in Figure 4.4,  $\theta$  is defined as  $\theta = \{\mathbf{W}^{11}, \mathbf{W}^{12}, \mathbf{W}^2, \mathbf{w}, \mathbf{b}^1, \mathbf{b}^2, b\}$  and correspondingly we define  $\mathfrak{R}(\theta)$  as:

$$\mathfrak{R}(\theta) = \|\mathbf{W}^{11}\|_F^2 + \|\mathbf{W}^{12}\|_F^2 + \|\mathbf{W}^2\|_2^2 + \|\mathbf{w}\|_2^2 + \|\mathbf{b}^1\|_2^2 + \|\mathbf{b}^2\|_2^2 + b^2 \quad (4.6)$$

Note that we can also choose other regularizers for  $\theta$  such as those based on  $\ell_1$ -norm and we would like to leave it as one future work.

We now extend the 2 hidden layer example to a  $N$  layer deep network. For a  $N$  layer deep network, the parameters are  $\mathbf{X}, \mathbf{x}_0$  and  $\theta = \{\mathbf{W}^{11}, \mathbf{W}^{12}, \mathbf{W}^2, \dots, \mathbf{W}^N, \mathbf{b}^1, \dots, \mathbf{b}^N, \mathbf{w}, b\}$  where  $\mathbf{W}^n$  are the weights for the  $n$ -th layer and  $\mathbf{b}^n$  is the bias for  $n$ -th layer with  $1 < n \leq N$ . The input to the first hidden layer is triplet  $(v_i, v_j, v_k)$ , i.e.,  $\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k$ . And the input to the  $n$ -th layer,  $1 < n \leq N$ , is the output of the  $(n-1)$ -th layer, i.e.,  $\mathbf{z}^{(n-1)1}$  and  $\mathbf{z}^{(n-1)2}$ . The output of the first layer is given by Eq. (4.4) and the output of the  $n$ -th layer,  $1 < n < N$  is given as:

$$\mathbf{z}^{n1} = \tanh(\mathbf{W}^n \mathbf{z}^{(n-1)1} + \mathbf{b}^n), \quad \mathbf{z}^{n2} = \tanh(\mathbf{W}^n \mathbf{z}^{(n-1)2} + \mathbf{b}^n) \quad (4.7)$$

And the output of the  $N$ -th layer is given as

$$f(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\mathbf{w}^T \mathbf{z}^{N1} + b), \quad f(\mathbf{x}_i, \mathbf{x}_k) = \tanh(\mathbf{w}^T \mathbf{z}^{N2} + b) \quad (4.8)$$

#### 4.4 Optimization of SiNE

Following the common way, we employ the backpropagation to optimize the deep network for SiNE. The key idea of backpropagation is to update the parameters in a backward direction by propagating "errors" backward to efficiently calculate the gradients. Basically, we want to optimize Eq. (4.3) w.r.t to  $\mathbf{X}, \mathbf{x}_0$  and  $\theta$ . The key step of optimizing Eq. (4.3) is to get the gradient of  $\max(0, f(\mathbf{x}_i, \mathbf{x}_k) + \delta - f(\mathbf{x}_i, \mathbf{x}_j))$  and  $\max(0, f(\mathbf{x}_i, \mathbf{x}_0) + \delta - f(\mathbf{x}_i, \mathbf{x}_j))$  with respect to the parameters,  $\mathbf{X}, \mathbf{x}_0$  and  $\theta$ . With

the gradient, we then can update the parameters using gradient descent method. The details of how to derive the derivatives for backpropagation can be found in Appendix <sup>3</sup>.

#### 4.5 Training SiNE

We train SiNE based on mini-batch stochastic gradient descent with respect to the parameters of the deep network, i.e.  $\theta$ , the signed network embedding  $\mathbf{X}$  and the virtual node embedding  $\mathbf{x}_0$ . It is well known that for signed social networks in social media, the number of links of nodes follows power-law distributions, i.e., many nodes have only a small number of links while only a small number of nodes have a large number of links. This will cause some nodes to have a large number of training triplets. To save computational cost, following the same idea used in word embedding [56], for a node that has a large number of training triplets, we randomly sample a subset of the training triplets for training. The size of the subset is chosen as  $S = 300$ . In other words, each node has at most 300 training triplets. The initialization of the parameters of the deep network follows the approach introduced in [24]. Specifically, we initialize the weights of hidden layer  $i$  by a uniform sampling from the interval  $[-\sqrt{\frac{6}{d_{i-1}+d_i}}, \sqrt{\frac{6}{d_{i-1}+d_i}}]$ , where  $d_{i-1}$  is number of units in the  $(i-1)$ -th layer and  $d_i$  is the number of units in the  $i$ -th layer. The signed network embedding  $\mathbf{X}$  is initialized as a zero matrix. The training algorithm for the proposed framework SiNE is summarized in Algorithm 2. From line 1 to line 9, we prepare the mini-batch training triplets. In line 10, we initialize the parameters of the deep network and the low-dimensional representations and we train the deep network from line 11 to line 24.

---

<sup>3</sup>Available on <http://www.public.asu.edu/~swang187/>

---

**Algorithm 2** Signed Network Embedding

---

**Require:** Signed social network  $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ ,  $d, \delta, \alpha$

**Ensure:** vector representation of nodes  $\mathbf{X}$

```
1: Initialize  $\mathcal{P}$  and  $\mathcal{P}_0$  as  $\mathcal{P} = \emptyset$  and  $\mathcal{P}_0 = \emptyset$ 
2: for  $i=1:n$  do
3:   if  $v_i$  whose 2-hop networks have only positive links then
4:     extract triplets with virtual nodes and put them into  $\mathcal{P}_0$  (sample some if
       necessary)
5:   else
6:     extract triplets and put them in  $\mathcal{P}$  (sample some if necessary)
7:   end if
8: end for
9: prepare mini-batch from  $\mathcal{P}$  and  $\mathcal{P}_0$ 
10: initialize the parameters of the deep network and signed network embedding
11: repeat
12:   for each mini-batch do
13:     Forward propagation
14:     for  $n = 1:N$  do
15:       calculate  $\mathbf{z}^{n1}, \mathbf{z}^{n2}$ 
16:     end for
17:     Backpropagation
18:     Update  $\mathbf{w}$  and  $b$ 
19:     for  $n= N:1$  do
20:       update  $\mathbf{W}^n, \mathbf{b}^n$  (or  $\mathbf{W}^{11}, \mathbf{W}^{12}$  if  $n = 1$ )
21:     end for
22:     update related  $\mathbf{X}$  and  $\mathbf{x}_0$ 
23:   end for
24: until Convergence
25: return  $\mathbf{X}$ 
```

---

### 4.5.1 Time Complexity

Let  $d$  be the dimension of the embedding and  $d_n, 1 \leq n \leq N$ , be the number of nodes in the  $n$ -th layer of the deep network. For a triplet, the computational cost of forward propagation in the  $n$ -th layer, i.e., the computation of  $\mathbf{z}^{n1}, \mathbf{z}^{n2}$ , is  $\mathcal{O}(d_{i-1}d_i)$  and the computational cost of back-propagation in the  $n$ -th layer, i.e. the computation of  $\frac{\partial f(\mathbf{x}_i, \mathbf{x}_k)}{\partial \mathbf{W}^n}$ , is also  $\mathcal{O}(d_{i-1}d_i)$ . Thus, the cost of forward and backward propagation for one triplet is  $\mathcal{O}(dd_1 + \sum_{n=1}^N d_{i-1}d_i)$ . Since for each node, we sample no more than  $S = 300$  training triplets, the total number of triplets for training, i.e.,  $C = |\mathcal{P}| + |\mathcal{P}|_0$ , is approximately  $\mathcal{O}(m \cdot S)$ , where  $m$  is number of nodes. Thus, the overall computational cost for training SiNE is  $\mathcal{O}(tmS(dd_1 + \sum_{n=1}^N d_{i-1}d_i))$ , where  $t$  is number of epochs it takes to converge. And in our experiments,  $t$  is about 100 for the datasets used.

## 4.6 Extending SiNE for Attributed Signed Social Network

The proposed framework is designed for signed network without attributes. However, it is also a flexible model that can be extended for simultaneously handling both attributes and signed network [90]. One simple way is to concatenate the node attributes with the learned node embedding as the new node representation. However, such simple concatenation may not be optimal as the information contained in node attributes and signed network structure are not well fused. Thus, we propose to modify SiNE to learn embeddings that model both node attributes and signed network. To do this, we first need to investigate the attributes similarity of two positively or negatively linked users in signed social networks, which serves as a foundation to extend SiNE.

The data analysis of the attributes similarity between two positively or negatively

linked users in signed social networks are as follows. Let  $p_i$  and  $n_i$  denote the number of positive and negative links of  $v_i$ , respectively. We construct two sets for each user  $u_i$  with the same size of  $\min(p_i, n_i)$ . These sets correspond to (i) a friend circle  $\mathcal{P}_i$  including randomly selected users who have positive links with  $v_i$ ; and (ii) a foe circle  $\mathcal{N}_i$  containing randomly selected users who have negative links with  $v_i$ . We then create the positive link set  $\mathcal{P}$  and negative link set  $\mathcal{N}$  as

$$\begin{aligned}\mathcal{P} &= \{(v_i, v_k) | v_k \in \mathcal{P}_i, i = 1, \dots, n\} \\ \mathcal{N} &= \{(v_i, v_k) | v_k \in \mathcal{N}_i, i = 1, \dots, n\}\end{aligned}\tag{4.9}$$

With these two sets, we can then calculate the similarity for each pair of users  $(v_i, v_k)$  in  $\mathcal{P}$  and  $\mathcal{N}$ . We investigate two ways of calculating similarity as follows

- CA: For a pair of users,  $(v_i, v_k)$ , we compute the similarity  $sim(v_i, v_k)$  as the number of common attributes by both  $v_i$  and  $v_k$ ; and
- COSINE: We compute  $sim(v_i, v_k)$  as the cosine similarity between the attributes of  $v_i$  and attributes of  $v_k$

Let  $\mathbf{s}_p \in \mathbb{R}^{|\mathcal{P}| \times 1}$  denote the similarity vector of each pair of users in  $\mathcal{P}$  and  $\mathbf{s}_n$  denote the similarity vectors for  $\mathcal{N}$ . The mean values of  $\mathbf{s}_p$  and  $\mathbf{s}_n$  are shown in Table 4.1. From the table, we observe that users are likely to have more similar attributes with their friends than their foes. To statistically verify the observations, we conduct a two-sample t-test.

For two vectors  $\{\mathbf{x}, \mathbf{y}\}$ , the null hypothesis  $H_0$  and the alternative hypothesis  $H_1$  of the two-sample  $t$ -test are defined as follows:

$$H_0 : \mathbf{x} \leq \mathbf{y} \quad H_1 : \mathbf{x} > \mathbf{y}\tag{4.10}$$

where the null hypothesis indicates that the mean of  $\mathbf{x}$  is less than or equal to that of  $\mathbf{y}$ . We perform the t-test on  $\{\mathbf{s}^p, \mathbf{s}^n\}$  to substantiate the aforementioned observation. The



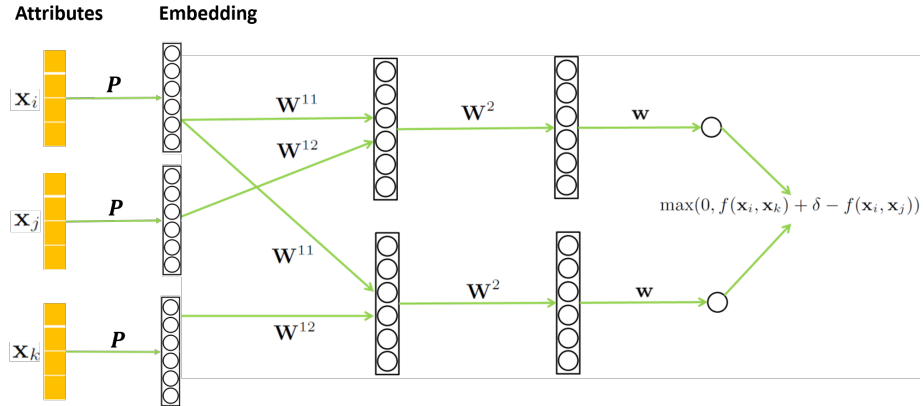
	Epinions		Slashdot	
	CA	COSINE	CA	COSINE
$\mathcal{P}$	75.93	0.0650	21.24	0.0332
$\mathcal{N}$	67.16	0.0540	16.64	0.0289

**Table 4.1:** Average User Attributes Similarities in  $\mathcal{P}$  and  $\mathcal{N}$

	Epinions		Slashdot	
	CA	COSINE	CA	COSINE
$\{\mathbf{s}^p, \mathbf{s}^n\}$	2.31e-31	7.72e-110	3.27e-144	7.85e-148

**Table 4.2:** P-value of t-test Results

null hypothesis is that positively linked users have less common attributes than that of negatively linked users; therefore, if we reject the null hypothesis, then the assumption that positively linked have more common attributes than negatively linked users is verified. The null hypothesis is rejected at significance level  $\alpha = 0.01$  with  $p$ -value shown in Table 4.2, which verifies our observations statistically.



**Figure 4.5:** An Illustration of the Architecture of the Proposed SiNE for Attributed Signed Network.

This suggests that the principle for modeling both attributes and signed links is consistent with the principle used in SiNE, i.e., we should have our friends closer than our foes. Therefore, we can reuse the objective function of SiNE

$$\begin{aligned} \min_{\mathbf{x}_0, \theta} \quad & \frac{1}{C} \left[ \sum_{(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \in \mathcal{P}} \max(0, f(\mathbf{x}_i, \mathbf{x}_k) + \delta - f(\mathbf{x}_i, \mathbf{x}_j)) \right. \\ & \left. + \sum_{(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_0) \in \mathcal{P}_0} \max(0, f(\mathbf{x}_i, \mathbf{x}_0) + \delta_0 - f(\mathbf{x}_i, \mathbf{x}_j)) \right] + \alpha (\mathfrak{R}(\theta) + \|\mathbf{x}_0\|_2^2), \end{aligned} \quad (4.11)$$

where  $C = |\mathcal{P}| + |\mathcal{P}_0|$  is the size of the training data, and  $\theta$  is a set of parameters to define the similarity function  $f$ .  $\mathfrak{R}(\theta)$  is the regularizer to avoid overfitting and  $\alpha$  is a parameter to control the contribution of the regularizers. An illustration of  $f$  is shown in Figure 4.5. *Note that now  $\mathbf{X} \in \mathbb{R}^{d \times N}$  is the node attributes matrix with each column  $\mathbf{x}_i$  being the attributes of node  $v_i$ .* The attributes are first projected as  $g(\mathbf{P}\mathbf{x}_i + \mathbf{e})$  to form the node embedding, where  $\mathbf{P}$  is the weights,  $\mathbf{e}$  is the bias and  $g$  is activation function such as sigmoid, tanh or ReLu. The rest of the network are the same as the original SiNE.

## 4.7 Evaluating SiNE

In this subsection, we conduct experiments to evaluate the effectiveness of the proposed framework SiNE. We begin by introducing datasets. We then analyze the embedding learned by SiNE. To measure the quality of the embedding, we use the embedding for signed link perdition.

### 4.7.1 Datasets

The experiments are conducted on two real-world signed social network datasets, i.e., Epinions and Slashdot. Epinions is a popular product review site in which users can create both trust (positive) and distrust (negative) links to other users. Slashdot is a technology news platform where users can create friend (positive) and foe (negative)

links to other users. For both datasets, we filter out users who have no links, which leaves us 27,215 users for Epinions dataset and 33,407 users for Slashdot dataset. Some key statistics of the two datasets are summarized in Table 4.3. It is evident from the table that (1) both networks are very sparse; (2) positive links are denser than negative links.

Dataset	# users	# pos links	# neg links
Epinions	27,215	326,909	58,695
Slashdot	33,407	477,176	158,104

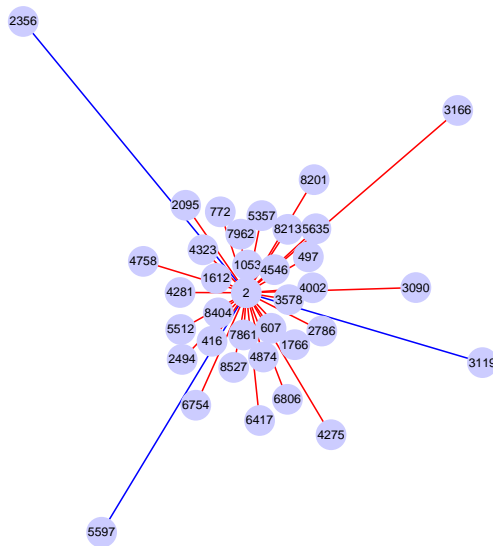
**Table 4.3:** Statistics of the Signed Network Datasets

#### 4.7.2 Analysis of the Signed Embedding

In this subsection, we would like to check whether the embedding learned by the deep learning framework SiNE can preserve the principle suggested by the extended structural balance theory - users are likely to be more similar to their friends than their foes. Specifically, we first train the model on the two datasets and learn the signed network embedding. In the experiment, we set  $d$  as 20,  $\alpha = 0.0001$  and  $N = 3$  with all the hidden layer dimension as 20. We will discuss the effects of  $d$  and  $N$  in detail later. Since we use  $\tanh$  as the activation function whose range is  $(-1, 1)$ , from Eq.(4.8), we have that  $f(\mathbf{x}_i, \mathbf{x}_j) \in (-1, 1)$  and  $f(\mathbf{x}_i, \mathbf{x}_k) \in (-1, 1)$ . In order to let  $f(\mathbf{x}_i, \mathbf{x}_j) \geq f(\mathbf{x}_i, \mathbf{x}_k) + \delta$  be valid,  $\delta$  should be within the range  $(0, 2)$ . The same holds for  $\delta_0$ . We empirically set  $\delta = 1$  and  $\delta_0 = 0.5$ . Then for each triplet  $(v_i, v_j, v_k)$  where  $v_i$  and  $v_j$  have a positive link while  $v_i$  and  $v_k$  have a negative link, we calculate the Euclidean distance for pairs of  $(v_i, v_j)$  and  $(v_i, v_k)$  and the average Euclidean distance (with standard deviation) are demonstrated in Table 4.4. From the table, we note

Dataset	Dis. to Friends (+)	Dis. to Foes (-)
Epinions	0.0584±0.0275	0.1195±0.0335
Slashdot	0.0538±0.0245	0.1028±0.0254

**Table 4.4:** Average Distance Between Users and Their Friends and Foes.



**Figure 4.6:** A Case Study of Signed Network Embedding.

that after embedding, nodes are indeed closer to their friends (positive link) than their foes (negative link), which suggests that the embedding from SiNE can perverse the principle suggested by extended structural balance theory.

A case study of the embedding distance between a user (or node 2) and his/her friends and foes is shown in Figure 4.6. The red lines denote positive links and the blue lines denote negative links. We use the length of the line to represent the embedding distance of two nodes. The longer the line is, the larger the embedding distance is. We observe from the figure that node 2 is likely to be closer to his/her friends than his/her foes.

### 4.7.3 Signed Link Prediction in Signed Social Networks

The learned signed network embedding can benefit various mining tasks of signed social networks. In this subsection, we check whether the learned signed network embedding can improve the performance of link prediction for signed social networks. For both datasets, we randomly select 80% links as training set and the remaining 20% as test set. We use the training set to learn the signed network embedding. With the learned signed network embedding, we train a logistic regression classifier on training dataset. Then we predict link on the test set with the logistic regression classifier. In real-world signed social networks such as Epinions and Slashdot, positive links are often much denser than negative links; hence positive and negative links are imbalanced in both training and testing sets. Therefore, following the common way to evaluate the signed link prediction problem [48, 4], we use AUC and F1 instead of accuracy to assess the performance. The random selection is carried out 5 times independently and the average AUC and F1 are reported in Table 4.5 and 4.6. The baseline methods in the tables are defined as:

- SC [45]: A spectral clustering algorithm is proposed where a signed version of Laplacian matrix is defined. In this experiment, for the link prediction purpose, we choose the top- $d$  eigen-vectors corresponding to the smallest eigenvalues of the signed Laplacian matrix as the low dimensional vector representations of nodes.
- FExtra [48]: This method extracts features from signed social networks. For each pair  $(v_i, v_j)$ , the extracted features include degree based and triad based features. Degree based features contain the degree information such as the number of incoming positive and negative links of  $v_i$ , the number of outgoing positive and negative links of  $v_j$  and so on. Triad based features include the

Dataset	SC	FExtra	MF	SiNE/ $\mathcal{P}_0$	SiNE
Epinions	0.8527	0.8626	0.8879	0.8845	<b>0.9242</b>
Slashdot	0.8495	0.8536	0.8725	0.8701	<b>0.8979</b>

**Table 4.5:** AUC Comparison of Signed Link Prediction on Epinions and Slashdot

Dataset	SC	FExtra	MF	SiNE/ $\mathcal{P}_0$	SiNE
Epinions	0.9089	0.9178	0.9343	0.9306	<b>0.9622</b>
Slashdot	0.8792	0.8839	0.8952	0.8924	<b>0.9149</b>

**Table 4.6:** F1 Comparison of Signed Link Prediction on Epinions and Slashdot

structure information of the triad that contains  $v_i$  and  $v_j$ .

- MF [37]: Matrix factorization based method which factorizes the adjacency matrix into two low rank latent matrices and predicts the links by the matrix reconstructed by the two low rank matrices.
- SiNE/ $\mathcal{P}_0$ : a variant of the proposed framework SiNE without considering virtual nodes. In other words, for SiNE/ $\mathcal{P}_0$ , we set  $\mathcal{P}_0 = \emptyset$  in Eq.(4.3).

For SC, FExtra and the proposed framework SiNE, we first obtain the new representations and then choose logistic regression as the basic classifier for a fair comparison. 5-fold cross validation is performed on the training set to select the parameters for SC, FExtra and MF. For SiNE, we empirically set  $d = 20$   $\delta = 1$ ,  $\delta_0 = 0.5$  and  $N = 3$  with all hidden layer dimension as 20. More details about parameters of SiNE will be discussed in the following subsection. From the Table 4.5 and 4.6, we make the following observations:

- The performance of the proposed framework SiNE is much better than FExtra. FExtra uses feature engineering to extract features manually; while SiNE learns

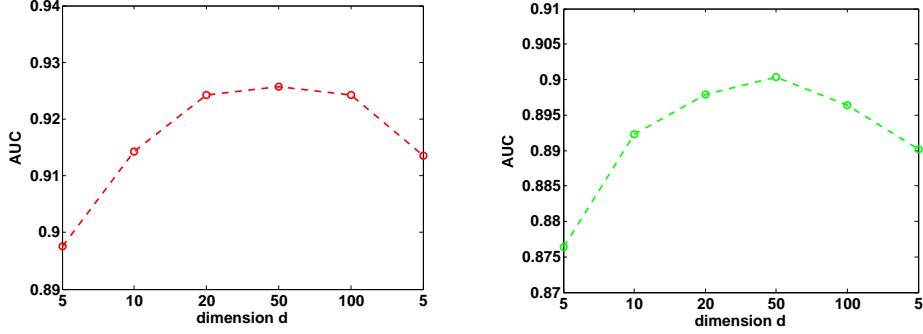
the representations from the data automatically. These results suggest that the representations learned by SiNE can greatly improve the performance of link prediction; and

- The performance of SiNE outperforms SC and MF. SC designs a signed Laplacian to preserve pair-wise relations of nodes, while SiNE preserves the principle suggested by the extended structural balance theory, which supports the capability of the objective function of signed network embedding.
- SiNE outperforms SiNE/ $\mathcal{P}_0$  because without considering the virtual node, the representation of nodes whose 2-hop networks have only negative links cannot be well trained. Since such nodes are few, the performance decrease is not much without considering virtual node.

We perform t-test on comparisons and it is evident from t-test that the improvement of SiNE compared to baseline methods is significant. In summary, the representations learned by the proposed framework SiNE can significantly improve the performance of link prediction in signed social networks.

#### 4.7.4 *Parameter Analysis*

In this subsection, we investigate the impact of embedding dimension  $d$ ,  $\delta$ ,  $\delta_0$  and number of layers  $N$  on the performance of link prediction. Throughout the experiments for parameter sensitivity analysis, we randomly select 80% links as training set and the remaining 20% as test set. The random selection is repeated 5 times and the average AUC will be reported.



(a) AUC on Epinions

(b) AUC on Slashdot

**Figure 4.7:** The Impact of Embedding Dimension  $d$  on SiNE for Signed Link Prediction

### Impact of $d$ :

To investigate the sensitivity of SiNE on  $d$ , we fix  $\delta = 1$ ,  $\delta_0 = 0.5$  and  $N = 3$ . We then vary  $d$  as  $\{5, 10, 20, 50, 100, 200\}$ . The average AUC for signed link prediction on both datasets are shown in figure 4.7(a) and 4.7(b), respectively. From the two figures, we note that with the increase of  $d$ , the signed link prediction performance first increases and then decreases after certain values. When  $d$  is small, we may lose too much information and embeddings do not have enough representation capacity. When  $d$  is large, the embedding tends to overfit. A value of  $d$  around 20 gives relatively good performance, which eases the parameter selection for  $d$ .

### Impact of $\delta$ and $\delta_0$ :

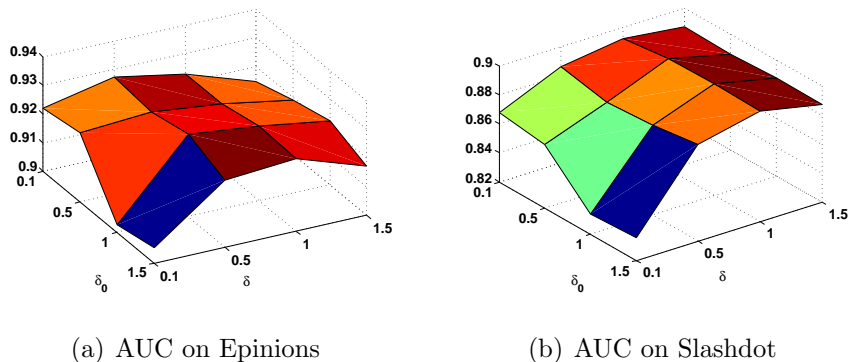
As shown in Eq.(4.3),  $\delta$  and  $\delta_0$  controls the similarity of a node with its friend and the node with its foe. To investigate the impact of  $\delta$  and  $\delta_0$ , we fix the dimension  $d$  to be 20 and  $N = 3$ . We then vary both  $\delta$  and  $\delta_0$  as  $\{0.1, 0.5, 1, 1.5\}$ . The results in terms of AUC under different combinations of  $\delta$  and  $\delta_0$  are shown in Figure 4.8. From the figure, we note that: (i) As the increase of  $\delta$ , the performance generally increases. This is because when  $\delta$  is large, we enforce a friend to sit closer to its friends and sit



Dataset	$N = 2$	$N = 3$	$N = 4$	$N = 5$	$N = 6$
Epinions	0.9124	0.9242	0.9278	0.9297	0.9254
Slashdot	0.8817	0.8979	0.9048	0.9044	0.9027

**Table 4.7:** AUC of SiNE on Signed Link Prediction with Different Number of Layers  $N$

more far away from his foes, which help us to learn high quality embedding for signed link prediction; and (ii) When  $\delta_0$  is large and  $\delta$  is small, e.g.,  $\delta_0 = 1.5$  and  $\delta = 0.1$ , the performance is relatively bad. A combination of  $(\delta_0, \delta)$  chosen from  $[0.5, 1]$  generally result in good embedding for signed link prediction.



**Figure 4.8:** The Impact of  $\delta$  and  $\delta_0$  on SiNE for Signed Link Prediction.

### Impact of $N$ :

To investigate the effects of  $N$ , we first fix  $d$  to be 20,  $\delta$  to be 1 and  $\delta_0$  to be 0.5. We then vary  $N$  as 2, 3, 4, 5, 6 with all the hidden dimensions as 20. The results in terms of AUC are reported in Table 4.7. From the table, we can see that as the network becomes deeper, the performance increases first then the increase become small, which suggests that by setting  $N = 2$  or  $N = 3$ , we can learn a relatively good embedding and at the same time save computational cost.

## DYNAMIC NETWORK EMBEDDING

In this chapter, we study dynamic network embedding. In many scenarios, networks are dynamic. For example, in a social network, existing users can change their preferences, add new friends and join new groups; and new users can join a social network and create new links. It is reported that 6 new profiles are created every second in Facebook <sup>1</sup>; and the number of Instagram users grows from 90 millions in Jan. 2013 to 800 millions in Sep. 2017 <sup>2</sup>.

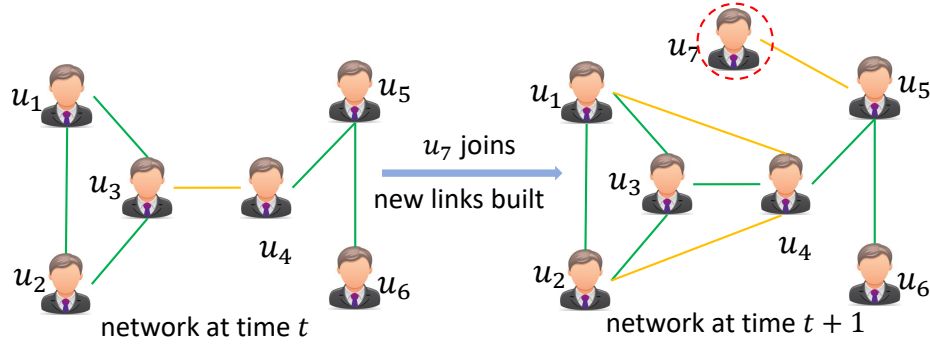
Dynamic networks present new challenges and opportunities for network representation learning because dynamic networks have unique characteristics that make static network embedding inapplicable to dynamic networks. *First*, a dynamic network contains temporal patterns that reveal the concept drift of nodes and can facilitate several network mining tasks such as next timestamp link prediction [1]. Figure 5.1 gives two snapshots of a dynamic social network at time  $t$  and  $t + 1$ , where a yellow line denotes a new link created within two consecutive timestamps. For example, in the left network means, the yellow line  $(u_3, u_4)$  means that the line is connected during  $(t - 1, t]$  while the black line  $(u_4, u_5)$  that the link is connected before  $t - 1$ . During time  $(t - 1, t]$ ,  $u_4$  becomes a friend of  $u_3$ , which implies that currently  $u_4$  is interested in  $u_3$  and thus it is likely for  $u_4$  to join  $u_3$ 's friend circle (befriending with  $u_1$  and  $u_2$ ) during  $(t, t + 1]$  as shown in the right network.  $u_4$  and  $u_6$  are friends of  $u_5$  respectively at time  $t$ , the fact that  $u_4$  and  $u_6$  are still not friends at time  $t + 1$  indicates the violation of balance theory, or they may not be friends in the near future. Therefore,

---

<sup>1</sup><http://www.pewresearch.org/fact-tank/2014/02/03/6-new-facts-about-facebook/>

<sup>2</sup>[www.statista.com/statistics/253577/number-of-monthly-active-instagram-users/](http://www.statista.com/statistics/253577/number-of-monthly-active-instagram-users/)

it is important to learn dynamic node latent features to capture temporal patterns; it would be sub-optimal to directly apply static network embedding by ignoring such temporal patterns. *Second*, in dynamic networks, new nodes can join and new links can be created. For example, in Figure 5.1,  $u_7$  joins the network and links  $(u_1, u_4)$ ,  $(u_2, u_4)$  are created during  $(t, t + 1]$ . It is important to learn embedding for new nodes and update the embeddings to reflect these changes; while static network embedding methods don't have an efficient way to achieve this. *Third*, the network size is unknown as new nodes can join while static network embedding usually assumes a fixed node set. Therefore, it is important to learn dynamic network embedding that can (i) capture temporal patterns of dynamic networks; and (ii) dynamically update/learn existing node representation when new links and new nodes are introduced. However, the work on dynamic network embedding is rather limited. The majority of existing dynamic network embedding algorithms [20, 97, 98] assume that the number of nodes is known and focus on updating node representations when new links are introduced using tensor decomposition [20] or temporal matrix factorization [97]. However, it is inflexible to assume that the node set is fixed as new nodes can join the network anytime. In addition, these approaches learn embeddings that are good at reconstructing the links, i.e., first-order proximity, however, recent advances in static network embedding have demonstrated that exploiting both first-order and second-order proximity can help learn better representations [79, 89]. Therefore, we investigate the challenging problem of dynamic network embedding where new nodes and new links can be introduced to the network anytime. Next, we formally define the problem followed



**Figure 5.1:** An Illustration of Dynamic Network

by the details of the proposed framework for dynamic network embedding.

### 5.1 Problem Formulation

Let  $\mathcal{G}_t = \{\mathcal{U}_t, \mathcal{E}_t\}$ ,  $t = 1, 2, \dots$ , be a snapshot of a dynamic network at time  $t$ , where  $\mathcal{U}_t = \{u_1, u_2, \dots, u_{n_t}\}$  is the set of  $n_t$  nodes at time  $t$  and  $\mathcal{E}_t \subset \mathcal{U}_t \times \mathcal{U}_t$  is the set of edges. Note that usually we have  $\mathcal{U}_t \subset \mathcal{U}_{t+1}$  as new nodes can join the network during  $(t, t + 1]$ . In addition, new links can also be introduced during  $(t, t + 1]$ , i.e.,  $\mathcal{E}_t \subset \mathcal{E}_{t+1}$ . We leave the deletion of links and nodes as one future work. We use  $\Delta\mathcal{G}_{t+1} = \{\Delta\mathcal{U}_{t+1}, \Delta\mathcal{E}_{t+1}\}$  to represent the set of new nodes and new links created during  $(t, t + 1]$ . For example, in Figure 5.1,  $\Delta\mathcal{U}_{t+1} = \{u_7\}$  and  $\Delta\mathcal{E}_{t+1} = \{e_{14}, e_{24}, e_{57}\}$ . With the aforementioned notations and definitions, the problem of dynamic network embedding can be formally stated as follows:

*Given a dynamic network  $\mathcal{G}$  with the new snapshot  $\mathcal{G}_t = \{\mathcal{U}_t, \mathcal{E}_t\}$  arriving at time  $t$ , we aim to learn the new representation  $\mathbf{U}_t \in \mathbb{R}^{d \times n_t}$  based on  $\Delta\mathcal{G}_t$ ,  $\mathcal{G}_{t-1}$  and  $\mathbf{U}_{t-1} \in \mathbb{R}^{d \times n_{t-1}}$  as*

$$f(\Delta\mathcal{G}_t, \mathcal{G}_{t-1}, \mathbf{U}_{t-1}) \rightarrow \mathbf{U}_t \quad (5.1)$$

*where  $f$  is the function we want to learn,  $d$  is the embedding dimension,  $n_t$  and  $n_{t-1}$  are the number of nodes at timestamps  $t$  and  $t - 1$ .*

## 5.2 Dynamic Network Embedding

In this section, we introduce the details of DNE. We will first provide a probabilistic model for static network embedding, which serves as a basic model for DNE.

### 5.2.1 Static Network Embedding

Recent advances in static network embedding have shown promising results by modeling first and second order proximity [79, 89]. Therefore, in this work, our basic static network embedding also explores the first-order and second-order proximity.

**Modeling First-Order Proximity** The first-order proximity is the pairwise proximity between vertexes. For any pair of vertexes, if  $e_{ij} = 1$ , there exists a positive first-order proximity between  $u_i$  and  $u_j$ . Otherwise, the first-order proximity between  $u_i$  and  $u_j$  is 0. The homophily theory [54] suggests that if two nodes are friend, then they are more likely to have common interests than two non-friend nodes. Thus, if two nodes are linked, they should have similar embedding; while for two non-linked nodes, they are less likely to share common properties. Therefore, we model the first-order proximity as:

$$\begin{aligned} p(e_{ij} = 1 | \mathbf{u}_i, \mathbf{u}_j) &= \sigma(\mathbf{u}_i^T \mathbf{u}_j), \\ p(e_{ij} = 0 | \mathbf{u}_i, \mathbf{u}_j) &= 1 - \sigma(\mathbf{u}_i^T \mathbf{u}_j) \end{aligned} \tag{5.2}$$

where  $\mathbf{u}_i \in \mathbb{R}^{d \times 1}$  is the first-order embedding of  $u_i$  and  $d$  is the embedding dimension.  $\sigma(x)$  is the sigmoid function defined as  $\sigma(x) = \frac{1}{1 + \exp(-x)}$ . By maximizing Eq.(5.2), we push two nodes with positive first-proximity to be close while two nodes without link to be far away. Let  $\mathbf{U}$  be the first-order embedding of the network with  $\mathbf{u}_i$  being the  $i$ -th column. With Eq.(5.2), the objective function for modeling first-order proximity is given as

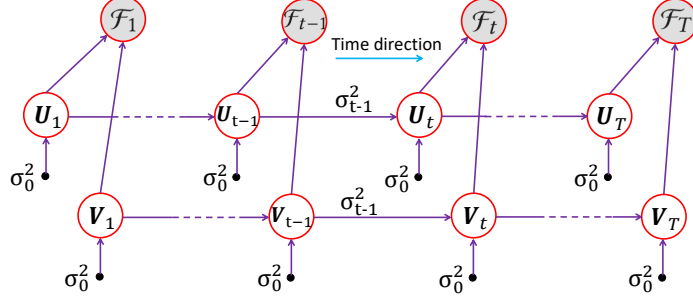
$$p(\mathcal{N}_i^+, \mathcal{N}_i^- | \mathbf{U}) = \prod_{j \in \mathcal{N}_i^+} \sigma(\mathbf{u}_i^T \mathbf{u}_j) \prod_{k \in \mathcal{N}_i^-} [1 - \sigma(\mathbf{u}_i^T \mathbf{u}_k)] \tag{5.3}$$

where  $\mathcal{N}_i^+$  is a set of nodes that have links with  $u_i$ ; while  $\mathcal{N}_i^-$  is a set of nodes that are not linked with  $u_i$ . However, in real-world social networks, the links observed are only a small proportion, with many others missing [51]. Thus,  $|\mathcal{N}_i^-|$  will be much larger than  $|\mathcal{N}_i^+|$ , which may result in the case that  $\mathcal{N}_i^-$  dominates the objective function. To avoid this effect, for each  $j \in \mathcal{N}_i^+$ , we randomly select  $r$  negative samples to build  $\mathcal{N}_i^-$ , where  $r$  is a positive integer we can tune.

**Modeling Second-Order Proximity** Modeling first-order proximity alone is not sufficient for preserving the network structure [79]. For example, it is very usual that two nodes are not linked because the network is sparse, but the two nodes can share many neighbors in common. These two nodes are likely to have similar properties because of their common neighbors. However, such proximity cannot be captured by Eq.(5.2) as their first-order proximity is 0. Therefore, it is important to seek an alternative notion of proximity that addresses the problem of sparsity. The second-order proximity between a pair of nodes describes the proximity of the pair’s neighborhood structure. For a pair  $(u_i, u_j)$ , the second-order proximity is determined by the similarity of  $\mathcal{N}_i^+$  and  $\mathcal{N}_j^+$ . We use  $w_{ij} = |\mathcal{N}_i^+ \cap \mathcal{N}_j^+|$  to calculate the second-order proximity, which counts the number of times  $u_i$  and  $u_j$  share a common neighbor. The probability that  $u_j$  appears  $w_{ij}$  times in the “contexts” of  $u_i$  can be given as

$$p(w_{ij}|\mathbf{u}_i, \mathbf{v}_j) = \sigma(\mathbf{u}_i^T \mathbf{v}_j)^{w_{ij}} \quad (5.4)$$

where  $\mathbf{v}_j \in \mathbb{R}^{d \times 1}$  is the second-order embedding of  $u_j$  and  $\mathbf{V}$  is the second-order embedding with  $\mathbf{v}_j$  being its  $j$ -th column. By maximizing Eq.(5.4), we enforce  $\mathbf{u}_i$  and  $\mathbf{v}_j$  to be close if  $w_{ij}$  is large. Obviously, the approach for modeling second-order proximity can be naturally extended to  $k$ -order proximity with a similar idea. In this work, we only consider first and second order proximity and leave the higher-order proximity as future work.



**Figure 5.2:** Graphical Illustration of DNE

Given the approach to model first-order and second-order proximity, the objective function for static network embedding is

$$\begin{aligned}
 p(\mathcal{G}|\mathbf{U}, \mathbf{V}) &= \left( \prod_{i=1}^N p(\mathcal{N}_i^+, \mathcal{N}_i^- | \mathbf{U}) \right) \left( \prod_{w_{ij} > 0} p(w_{ij} | \mathbf{u}_i, \mathbf{v}_j) \right) \\
 &= \prod_i \prod_{j \in \mathcal{N}_i^+} \sigma(\mathbf{u}_i^T \mathbf{u}_j) \prod_{k \in \mathcal{N}_i^-} [1 - \sigma(\mathbf{u}_i^T \mathbf{u}_k)] \prod_{j: w_{ij} > 0} \sigma(\mathbf{u}_i^T \mathbf{v}_j)^{w_{ij}}
 \end{aligned} \tag{5.5}$$

It is possible that some pairs of nodes share large amount of common friends, which result in large  $w_{ij}$  that dominates the objective function. To avoid this, for each node, we normalize the second-order proximity as  $w_{ij} \leftarrow \frac{w_{ij}}{\sum_j w_{ij}}$ .

### 5.2.2 Bayesian Dynamic Network Embedding

In this section, we extend the static network embedding to model dynamic networks. Let  $\mathbf{U}_t \in \mathbb{R}^{d \times n_t}$  and  $\mathbf{V}_t \in \mathbb{R}^{d \times n_t}$  denote the first and second order embedding at time  $t$ . In addition,  $\mathbf{u}_{i,t}$  and  $\mathbf{v}_{i,t}$  represent the  $i$ -th column of  $\mathbf{U}_t$  and  $\mathbf{V}_t$ , respectively

In dynamic networks, nodes change their latent positions gradually over time, which is generally called concept drift. To model the concept drift, we consider a diffusion process of the embedding vectors over time as

$$\begin{aligned}
 p(\mathbf{u}_{i,t+1} | \mathbf{u}_{i,t}) &\propto \mathcal{N}(\mathbf{u}_{i,t}, \sigma_t^2 \mathbf{I}) \mathcal{N}(0, \sigma_0^2) = \mathcal{N} \left( \frac{\mathbf{u}_{i,t}}{1 + \sigma_t^2 / \sigma_0^2}, \frac{1}{\sigma_t^{-2} + \sigma_0^{-2}} \mathbf{I} \right) \\
 p(\mathbf{v}_{i,t+1} | \mathbf{v}_{i,t}) &\propto \mathcal{N}(\mathbf{v}_{i,t}, \sigma_t^2 \mathbf{I}) \mathcal{N}(0, \sigma_0^2) = \mathcal{N} \left( \frac{\mathbf{v}_{i,t}}{1 + \sigma_t^2 / \sigma_0^2}, \frac{1}{\sigma_t^{-2} + \sigma_0^{-2}} \mathbf{I} \right)
 \end{aligned}$$

The basic idea is that  $\mathbf{u}_{i,t+1}$  is dependent on  $\mathbf{u}_{i,t}$  by the Gaussian distribution  $\mathcal{N}(\mathbf{u}_{i,t}, \sigma_t^2 \mathbf{I})$ , where the variance  $\sigma_t^2$  of the transition kernel is given as

$$\sigma_t^2 = D(\tau_{t+1} - \tau_t) \quad (5.6)$$

$D$  is a global diffusion constant and  $(\tau_{t+1} - \tau_t)$  is the time between subsequent observations [8]. Thus, a larger  $(\tau_{t+1} - \tau_t)$  can result in larger  $\sigma_t^2$ , which means the latent features can diffuse more. However, it is possible that  $\sigma_t^2$  is too large, which can result in large  $\mathbf{u}_{i,t+1}$ . To guarantee the smoothness, at every time step  $t$ , we also add an additional Gaussian prior with zero mean and variance  $\sigma_0^2$  which prevents the embedding vectors from growing too large [3]. The effect of  $\mathcal{N}(0, \sigma_0^2)$  is to drag  $\mathbf{u}_{i,t+1}$  to origin so that  $\mathbf{u}_{i,t+1}$  will not become too large. Specifically, when  $\sigma_t^2$  is small, i.e.,  $\sigma_0^2 \gg \sigma_t^2$ , the damping to the origin is very weak; when  $\sigma_t^2$  becomes larger, i.e.,  $\sigma_t^2$  is close to or larger than  $\sigma_0^2$ , the damping effect becomes strong, which can avoid the embedding becomes too large.

At time  $t = 1$ , all nodes are treated as new nodes and we define  $p(\mathbf{u}_{i,1}|\mathbf{u}_0)$  and  $p(\mathbf{v}_{i,1}|\mathbf{v}_0)$  as

$$\begin{aligned} p(\mathbf{u}_{i,1}|\mathbf{u}_{i,0}) &= p(\mathbf{u}_{i,1}) \equiv \mathcal{N}(\mathbf{0}, \sigma_0^2 \mathbf{I}), \\ p(\mathbf{v}_{i,1}|\mathbf{v}_{i,0}) &= p(\mathbf{v}_{i,1}) \equiv \mathcal{N}(\mathbf{0}, \sigma_0^2 \mathbf{I}) \end{aligned}$$

Similarly, when a new node  $u_k$  joins during time  $(t - 1, t]$ , we have

$$p(\mathbf{u}_{k,t}) \equiv \mathcal{N}(\mathbf{0}, \sigma_0^2 \mathbf{I}), \quad p(\mathbf{v}_{k,t}) \equiv \mathcal{N}(\mathbf{0}, \sigma_0^2 \mathbf{I}) \quad (5.7)$$

We further assume that given  $\mathbf{U}_t$  and  $\mathbf{V}_t$ ,  $\mathbf{U}_{t+1}$  is conditionally independent with  $\mathbf{V}_{t+1}$ , i.e.,

$$\begin{aligned} p(\mathbf{U}_{t+1}, \mathbf{V}_{t+1}|\mathbf{U}_t, \mathbf{V}_t) &= p(\mathbf{U}_{t+1}|\mathbf{U}_t)p(\mathbf{V}_{t+1}|\mathbf{V}_t) \\ &= \prod p(\mathbf{u}_{i,t+1}|\mathbf{u}_{i,t})p(\mathbf{v}_{i,t+1}|\mathbf{v}_{i,t}) \end{aligned} \quad (5.8)$$



Given the above assumptions, our joint distribution factorizes as

$$\begin{aligned}
& p(\mathcal{F}_{1:T}, \mathbf{U}_{1:T}, \mathbf{V}_{1:T}) \tag{5.9} \\
&= \prod_{t=0}^{T-1} p(\mathbf{U}_{t+1}, \mathbf{V}_{t+1} | \mathbf{U}_t, \mathbf{V}_t) \prod_{t=1}^T p(\mathcal{F}_t | \mathbf{U}_t, \mathbf{V}_t) \\
&= \left[ \prod_{t=0}^{T-1} p(\mathbf{U}_{t+1} | \mathbf{U}_t) p(\mathbf{V}_{t+1} | \mathbf{V}_t) \right] \left[ \prod_{t=1}^T \prod_{i=1}^{n_{t_i}} p(\mathcal{N}_{i,t}^+, \mathcal{N}_{i,t}^- | \mathbf{U}_t) \prod_{w_{ij,t} > 0} p(w_{ij,t} | \mathbf{u}_{i,t}, \mathbf{v}_{j,t}) \right] \tag{5.10}
\end{aligned}$$

where  $\mathcal{F}_{1:T} = \{\mathcal{F}_1, \dots, \mathcal{F}_T\}$  represents the set of observed data from time 1 to  $T$  with  $\mathcal{F}_t = \{\mathcal{N}_t^+, \mathcal{N}_t^-, \mathcal{W}_t\}$ . Here  $\mathcal{N}_t^+ = \{\mathcal{N}_{t,i}^+, i = 1, \dots, n_t\}$ ,  $\mathcal{N}_t^- = \{\mathcal{N}_{t,i}^-, i = 1, \dots, n_t\}$  and  $\mathcal{W}_t = \{\mathcal{W}_{t,i}, i = 1, \dots, n_t\}$ . Note that  $\mathcal{N}_{t,i}^+$ ,  $\mathcal{N}_{t,i}^-$  and  $\mathcal{W}_{t,i}$  are the first and second-order proximity of  $u_i$  at time  $t$  which are built based on  $\Delta\mathcal{G}_t$  and  $\mathcal{G}_{t-1}$ . A graphical representation of DNE is shown in Figure 5.2.

**Discussion** It is noteworthy that DNE can dynamically update existing nodes' latent features when new links are introduced and can also learn new nodes' representations. When an existing node  $u_i$  creates new links, DNE moves  $u_i$ 's latent representation towards the new position by modeling the first and second-order proximity with the new observations. Meanwhile, DNE uses a latent diffusion process to make sure that  $u_i$ 's latent features don't drift too much so as to keep partial past information. Similarly, when a new nodes  $u_k$  is introduced, DNE samples the latent representation of the new node from a prior distribution and then move the representation by modeling its first-order and second-order proximity.

### 5.3 Parameter Inference

In this section, we introduce details of parameter inference. We are interested in maximizing the posterior distribution over parameters conditioned on the observa-

tions as

$$p(\mathbf{U}_{1:T}, \mathbf{V}_{1:T} | \mathcal{F}_{1:T}) = \frac{p(\mathbf{U}_{1:T}, \mathbf{V}_{1:T}, \mathcal{F}_{1:T})}{\int p(\mathbf{U}_{1:T}, \mathbf{V}_{1:T} | \mathcal{F}) d\mathbf{U}_{1:T} d\mathbf{V}_{1:T}} \quad (5.11)$$

However, the denominator of the above equation is intractable. Therefore, we adopt variational inference [9] to tackle this problem.

### 5.3.1 Update Rules

The essential idea of variation inference is to approximate the posterior with a simpler variational distribution  $q_\theta(\mathbf{U}_{1:t}, \mathbf{V}_{1:t})$  by minimizing the Kullback-Leibler (KL) divergence to the posterior, where  $\theta$  is the set of the parameters of the variational distribution, which will be explained in detail below. Minimizing the KL divergence is equivalent to optimizing the evidence lower bound (ELBO), which is given as

$$\mathcal{L}(\theta) = \mathbb{E}_{q_\theta} [\log p(\mathbf{U}_{1:T}, \mathbf{V}_{1:T}, \mathcal{F}_{1:t})] - \mathbb{E}_{q_\theta} [\log q_\theta(\mathbf{U}_{1:T}, \mathbf{V}_{1:T})] \quad (5.12)$$

In dynamic networks, the data arrives sequentially. Thus, we can only condition our model on the past instead of the future observations. The inference algorithm needs to iteratively update the variational distribution  $q_\theta$  as evidence from each time step  $t$  becomes available. Thus, we use a variational distribution that factorizes across all times, i.e.,

$$q_\theta(\mathbf{U}_{1:T}, \mathbf{V}_{1:T}) = \prod_{t=1}^T q_{\theta_t}(\mathbf{U}_t, \mathbf{V}_t) \quad (5.13)$$

With the above simplification, we can update the variational factor at a given time  $t$  based on the evidence at time  $t$  and the approximate posterior of the previous time step. Furthermore, at every time  $t$ , we use a fully-factorized distribution as

$$q_{\theta_t}(\mathbf{U}_t, \mathbf{V}_t) = \prod_{i=1}^{n_t} \mathcal{N}(\mathbf{u}_{i,t}; \boldsymbol{\mu}_{ui,t}, \boldsymbol{\Sigma}_{ui,t}) \mathcal{N}(\mathbf{v}_{i,t}; \boldsymbol{\mu}_{vi,t}, \boldsymbol{\Sigma}_{vi,t}) \quad (5.14)$$

where  $\theta_t = \{\boldsymbol{\mu}_{ui,t}, \boldsymbol{\Sigma}_{ui,t}, \boldsymbol{\mu}_{vi,t}, \boldsymbol{\Sigma}_{vi,t}, i = 1, \dots, n_t\}$ .  $\boldsymbol{\mu}_{ui,t}$  and  $\boldsymbol{\mu}_{vi,t}$  are the means and  $\boldsymbol{\Sigma}_{ui,t} = \text{diag}(\mathbf{s}_{ui,t}^2)$  and  $\boldsymbol{\Sigma}_{vi,t} = \text{diag}(\mathbf{s}_{vi,t}^2)$  are diagonal matrices with  $\mathbf{s}_{ui,t}^2$  and  $\mathbf{s}_{vi,t}^2$  as

the on diagonal elements. We now describe how we sequentially compute  $q_{\theta_t}(\mathbf{U}_t, \mathbf{V}_t)$  and use the result to proceed to the next time step. Following other Markovian dynamical systems [3], we assume following recursion

$$p(\mathbf{U}_t, \mathbf{V}_t | \mathcal{F}_{1:t}) \propto p(\mathcal{F}_t | \mathbf{U}_t, \mathbf{V}_t) p(\mathbf{U}_t, \mathbf{V}_t | \mathcal{F}_{1:t-1}) \quad (5.15)$$

With the above assumption, the ELBO in Eq.(5.12) can be written as  $\mathcal{L}(\theta) = \sum_{t=1}^T \mathcal{L}_t$ , where  $\mathcal{L}_t$  is given as

$$\mathcal{L}_t = \mathbb{E}_{q_{\theta_t}}[\log p(\mathcal{F}_t | \mathbf{U}_t, \mathbf{V}_t)] + \mathbb{E}_{q_{\theta_t}}[\log p(\mathbf{U}_t, \mathbf{V}_t | \mathcal{F}_{1:t-1})] - \mathbb{E}_{q_{\theta_t}}[\log q_{\theta_t}(\mathbf{U}_t, \mathbf{V}_t)] \quad (5.16)$$

The entropy  $-\mathbb{E}_{q_{\theta_t}}[\log q(\mathbf{U}_t, \mathbf{V}_t)]$  can be analytically calculated as

$$\begin{aligned} -\mathbb{E}_{q_{\theta_t}}[\log q(\mathbf{U}_t, \mathbf{V}_t)] &= - \int q_{\theta_t}(\mathbf{U}_t, \mathbf{V}_t) \log q_{\theta_t}(\mathbf{U}_t, \mathbf{V}_t) d\mathbf{U}_t d\mathbf{V}_t \\ &= \sum_i \sum_{k=1}^K (\log \mathbf{s}_{ui,t}(k) + \log \mathbf{s}_{vi,t}(k)) + n_t K (1 + \log 2\pi) \end{aligned} \quad (5.17)$$

where  $\mathbf{s}_{ui,t}(k)$  means the  $k$ -the element of  $\mathbf{s}_{ui,t}$ . To tackle the second term of  $\mathcal{L}_t$ , we need to consider two cases, i.e., new nodes joined at time  $t$  and existing nodes. For a new node  $u_i$  joined at  $t$ , it is independent of  $\mathcal{F}_{1:t-1}$ . Then  $p(\mathbf{u}_{i,t}, \mathbf{v}_{i,t} | \mathcal{F}_{1:t-1})$  reduces to

$$p(\mathbf{u}_{i,t}, \mathbf{v}_{i,t} | \mathcal{F}_{1:t-1}) = p(\mathbf{u}_{i,t}, \mathbf{v}_{i,t}) = \mathcal{N}(\mathbf{0}, \sigma_0^2 \mathbf{I}) \mathcal{N}(\mathbf{0}, \sigma_0^2 \mathbf{I}) \quad (5.18)$$

For existing nodes, directly calculating  $p(\mathbf{u}_{i,t}, \mathbf{v}_{i,t} | \mathcal{F}_{1:t-1})$  is intractable. Since  $p(\mathbf{u}_{t-1}, \mathbf{v}_{t-1} | \mathcal{F}_{1:t-1}) \approx q(\mathbf{u}_{t-1}, \mathbf{v}_{t-1})$ , we can approximate  $p(\mathbf{u}_t, \mathbf{v}_t | \mathcal{F}_{1:t-1})$  as

$$\begin{aligned} p(\mathbf{u}_{i,t}, \mathbf{v}_{i,t} | \mathcal{F}_{1:t-1}) &= \mathbb{E}_{p(\mathbf{u}_{i,t-1}, \mathbf{v}_{i,t-1} | \mathcal{F}_{1:t-1})} [p(\mathbf{u}_{i,t}, \mathbf{v}_{i,t} | \mathbf{u}_{i,t-1}, \mathbf{v}_{i,t-1})] \\ &\approx \mathbb{E}_{q_{\theta_t}(\mathbf{u}_{i,t-1}, \mathbf{v}_{i,t-1})} [p(\mathbf{u}_{i,t}, \mathbf{v}_{i,t} | \mathbf{u}_{i,t-1}, \mathbf{v}_{i,t-1})] \end{aligned} \quad (5.19)$$

With  $q_{\theta_t}(\mathbf{u}_{i,t-1}, \mathbf{v}_{i,t-1})$  given in Eq.(5.14),  $p(\mathbf{u}_{i,t}, \mathbf{v}_{i,t} | \mathbf{u}_{i,t-1}, \mathbf{v}_{i,t-1})$  given in Eq.(5.8) and the above approximation, we can get that

$$p(\mathbf{u}_{i,t}, \mathbf{v}_{i,t} | \mathcal{F}_{1:t-1}) \approx \mathcal{N}(\mathbf{u}_{i,t}; \tilde{\boldsymbol{\mu}}_{ui,t}, \tilde{\boldsymbol{\Sigma}}_{ui,t}) \mathcal{N}(\mathbf{v}_{i,t}; \tilde{\boldsymbol{\mu}}_{vi,t}, \tilde{\boldsymbol{\Sigma}}_{vi,t}) \quad (5.20)$$

where  $\tilde{\boldsymbol{\mu}}_{ui,t}$  and  $\tilde{\boldsymbol{\Sigma}}_{ui,t}$  are given as

$$\begin{aligned}\tilde{\boldsymbol{\mu}}_{ui,t} &= \tilde{\boldsymbol{\Sigma}}_{ui,t}(\boldsymbol{\Sigma}_{ui,t-1} + \sigma_t^2 \mathbf{I})^{-1} \boldsymbol{\mu}_{ui,t-1} \\ \tilde{\boldsymbol{\Sigma}}_{ui,t} &= [(\boldsymbol{\Sigma}_{ui,t-1} + \sigma_t^2 \mathbf{I})^{-1} + (1/\sigma_0^2) \mathbf{I}]^{-1}\end{aligned}\tag{5.21}$$

Note that  $\boldsymbol{\Sigma}_{ui,t}$  is a diagonal matrix and thus the calculation of  $\tilde{\boldsymbol{\mu}}_{ui,t}$  and  $\tilde{\boldsymbol{\Sigma}}_{ui,t}$  is very efficient. In addition, the resulting matrix  $\tilde{\boldsymbol{\Sigma}}_{ui,t}$  is also diagonal. For simplicity of notation, these two cases can be written as

$$\begin{aligned}p(\mathbf{u}_{i,t}, \mathbf{v}_{i,t} | \mathcal{F}_{1:t-1}) &\approx \mathcal{N}(\mathbf{u}_{i,t}; \tilde{\boldsymbol{\mu}}_{ui,t}, \tilde{\boldsymbol{\Sigma}}_{ui,t}) \mathcal{N}(\mathbf{v}_{i,t}; \tilde{\boldsymbol{\mu}}_{vi,t}, \tilde{\boldsymbol{\Sigma}}_{vi,t}) \\ \tilde{\boldsymbol{\mu}}_{ui,t} &= \begin{cases} \tilde{\boldsymbol{\Sigma}}_{ui,t}(\boldsymbol{\Sigma}_{ui,t-1} + \sigma_t^2 \mathbf{I})^{-1} \boldsymbol{\mu}_{ui,t-1}, & u_i \text{ joins before } t-1 \\ \mathbf{0}, & u_i \text{ joins at } (t-1, t] \end{cases} \\ \tilde{\boldsymbol{\Sigma}}_{ui,t} &= \begin{cases} [(\boldsymbol{\Sigma}_{ui,t-1} + \sigma_t^2 \mathbf{I})^{-1} + (1/\sigma_0^2) \mathbf{I}]^{-1}, & u_i \text{ joins before } t-1 \\ \sigma_0^2 \mathbf{I}, & u_i \text{ joins at } (t-1, t] \end{cases}\end{aligned}\tag{5.22}$$

The calculation of  $\tilde{\boldsymbol{\mu}}_{vi,t}$  and  $\tilde{\boldsymbol{\Sigma}}_{vi,t}$  are the same as that in Eq.(5.22) by replacing the subscript  $u$  by  $v$  and we omit the detail here. With this approximation, the second term can be written as

$$\begin{aligned}&\mathbb{E}_{q_{\theta_t}}[\log p(\mathbf{U}_t, \mathbf{V}_t) | \mathcal{F}_{1:t-1}] \\ &\approx - \sum_i \sum_{k=1}^K [\log \tilde{\mathbf{s}}_{ui,t}(k) + \frac{\mathbf{s}_{ui,t}(k)^2 + (\mu_{ui,t}(k) - \tilde{\mu}_{ui,t}(k))^2}{2\tilde{\mathbf{s}}_{ui,t}(k, k)^2}] \\ &\quad - \sum_i \sum_{k=1}^K [\log \tilde{\mathbf{s}}_{vi,t}(k, k) + \frac{\mathbf{s}_{vi,t}(k, k)^2 + (\mu_{vi,t}(k) - \tilde{\mu}_{vi,t}(k))^2}{2\tilde{\mathbf{s}}_{vi,t}(k, k)^2}] - n_t K (1 + \log 2\pi)\end{aligned}\tag{5.23}$$

Obtaining a closed form format of the first term in  $\mathcal{L}_t$  is intractable. Thus, we estimate the gradient of the first term in  $\mathcal{L}_t$  by sampling from the variational distribution. We use reparameterization tricks to efficiently sample  $\mathbf{U}_t^{(s)}$  and  $\mathbf{V}_t^{(s)}$  [42]. The essential idea of the reparameterization trick for Gaussian distribution is that sampling from  $\mathcal{N}(\mu, \sigma)$  is equivalent to sampling a noise  $\epsilon$  from  $\mathcal{N}(0, 1)$  and then

represent the sample as  $z = \mu + \sigma\epsilon$ . It is easy to verify that  $z \sim \mathcal{N}(\mu, \sigma)$ . With reparameterization trick,  $\mathcal{L}_t$  becomes

$$\begin{aligned}\tilde{\mathcal{L}}_t &\approx \frac{1}{L} \sum_{l=1}^L \log p(\mathcal{F}_t | \mathbf{U}_t^{(l)}, \mathbf{V}_t^{(l)}) + \mathbb{E}_{q_{\theta_t}}[\log p(\mathbf{U}_t, \mathbf{V}_t) | \mathcal{F}_{1:t-1}] - \mathbb{E}_{q_{\theta_t}}[\log q_{\theta_t}(\mathbf{U}_t, \mathbf{V}_t)] \\ \mathbf{u}_{i,t}^{(l)} &= \boldsymbol{\mu}_{ui,t} + \mathbf{s}_{ui,t} \odot \boldsymbol{\epsilon}_{ui,t}^{(l)}, \quad \boldsymbol{\epsilon}_{ui,t}^{(l)} \sim \mathcal{N}(0, \mathbf{I}), \quad l = 1, \dots, L \\ \mathbf{v}_{i,t}^{(l)} &= \boldsymbol{\mu}_{vi,t} + \mathbf{s}_{vi,t} \odot \boldsymbol{\epsilon}_{vi,t}^{(l)}, \quad \boldsymbol{\epsilon}_{vi,t}^{(l)} \sim \mathcal{N}(0, \mathbf{I}), \quad i = 1, \dots, n_t\end{aligned}\tag{5.24}$$

where the second term and third term are given in Eq.(5.23) and Eq.(5.17), respectively. The reason we calculate closed form solution for the second and third term is to reduce gradient variance introduced by Monte Carlo sampling [42]. With Eq.(5.24), the gradient of  $\mathcal{L}_t$  w.r.t  $\boldsymbol{\mu}_{ui,t}$ ,  $\boldsymbol{\Sigma}_{ui,t}$ ,  $\boldsymbol{\mu}_{vi,t}$  and  $\boldsymbol{\Sigma}_{vi,t}$  are given as

$$\begin{aligned}\frac{\partial \tilde{\mathcal{L}}_t}{\partial \boldsymbol{\mu}_{ui,t}} &= -(\boldsymbol{\mu}_{ui,t} - \tilde{\boldsymbol{\mu}}_{ui,t}) \odot \tilde{\mathbf{s}}_{ui,t}^{-2} + \frac{1}{L} \sum_{l=1}^L \left[ \sum_{j \in \mathcal{N}_i^+} (1 - \sigma(\mathbf{u}_{i,t}^{(l)T} \mathbf{u}_{j,t}^{(l)})) \mathbf{u}_{j,t}^{(l)} \right. \\ &\quad \left. - \sum_{k \in \mathcal{N}_i^-} \sigma(\mathbf{u}_{i,t}^{(l)T} \mathbf{u}_{k,t}^{(l)}) \mathbf{u}_{k,t}^{(l)} + \sum_{j: w_{ij,t} > 0} w_{ij,t} (1 - \sigma(\mathbf{u}_i^{(l)T} \mathbf{v}_j^{(l)})) \mathbf{v}_j^{(l)} \right]\end{aligned}\tag{5.25}$$

$$\begin{aligned}\frac{\partial \tilde{\mathcal{L}}_t}{\partial \mathbf{s}_{ui,t}} &= \mathbf{s}_{ui,t}^{-1} - \mathbf{s}_{ui,t} \odot \tilde{\mathbf{s}}_{ui,t}^{-2} + \frac{1}{L} \sum_{l=1}^L \left[ \sum_{j \in \mathcal{N}_i^+} (1 - \sigma(\mathbf{u}_{i,t}^{(l)T} \mathbf{u}_{j,t}^{(l)})) \mathbf{u}_{j,t}^{(l)} \right. \\ &\quad \left. + \sum_{j: w_{ij,t} > 0} w_{ij,t} (1 - \sigma(\mathbf{u}_i^{(l)T} \mathbf{v}_j^{(l)})) \mathbf{v}_j^{(l)} - \sum_{k \in \mathcal{N}_i^-} \sigma(\mathbf{u}_{i,t}^{(l)T} \mathbf{u}_{k,t}^{(l)}) \mathbf{u}_{k,t}^{(l)} \right] \odot \boldsymbol{\epsilon}_{ui,t}^{(l)}\end{aligned}\tag{5.26}$$

$$\frac{\partial \tilde{\mathcal{L}}_t}{\partial \boldsymbol{\mu}_{vi,t}} = -(\boldsymbol{\mu}_{vi,t} - \tilde{\boldsymbol{\mu}}_{vi,t}) \odot \tilde{\mathbf{s}}_{vi,t}^{-2} + \frac{1}{L} \sum_{l=1}^L \sum_{j: w_{ji,t} > 0} w_{ji,t} (1 - \sigma(\mathbf{u}_j^{(l)T} \mathbf{v}_i^{(l)})) \mathbf{u}_j^{(l)}\tag{5.27}$$

$$\frac{\partial \tilde{\mathcal{L}}_t}{\partial \mathbf{s}_{vi,t}} = \mathbf{s}_{vi,t}^{-1} - \mathbf{s}_{vi,t} \odot \tilde{\mathbf{s}}_{vi,t}^{-2} + \frac{1}{L} \sum_{l=1}^L \left[ \sum_{j: w_{ji,t} > 0} w_{ji,t} (1 - \sigma(\mathbf{u}_j^{(l)T} \mathbf{v}_i^{(l)})) \mathbf{u}_j^{(l)} \right] \odot \boldsymbol{\epsilon}_{vi,t}^{(l)}\tag{5.28}$$

### 5.3.2 A Training Algorithm

With the updating rules given, we summarize the training algorithm of DNE in Algorithm 3. Next, we briefly review Algorithm 3. For each time  $t$ , we first initialize

$\boldsymbol{\mu}_{ui,t}$ ,  $\mathbf{s}_{ui,t}$ ,  $\boldsymbol{\mu}_{vi,t}$ , and  $\mathbf{s}_{vi,t}$  in Line 4. In Line 5, we calculate  $\tilde{\boldsymbol{\mu}}_{ui,t}$ ,  $\tilde{\mathbf{s}}_{ui,t}$ ,  $\tilde{\boldsymbol{\mu}}_{vi,t}$ , and  $\tilde{\mathbf{s}}_{vi,t}$  via Eq.(5.22) using  $\boldsymbol{\mu}_{ui,t-1}$ ,  $\mathbf{s}_{ui,t-1}$ ,  $\boldsymbol{\mu}_{vi,t-1}$ , and  $\mathbf{s}_{vi,t-1}$  learned in  $t - 1$ . From Lines 7 to 15, we update  $\boldsymbol{\mu}_{ui,t}$ ,  $\mathbf{s}_{ui,t}$ ,  $\boldsymbol{\mu}_{vi,t}$ , and  $\mathbf{s}_{vi,t}$  alternatively using reparameterization trick and gradient descent until convergence. After convergence, we can sample  $\mathbf{U}_t$  and  $\mathbf{V}_t$ . Note that at each time  $t$ , we only learn the embeddings based on previous embedding and the training data  $\mathcal{F}_t$ . No future information after  $t$  are used. This ensures that we can dynamically learn/update embeddings for time  $T + 1, T + 2, \dots$  when these training data are available.

### 5.3.3 Time Complexity Analysis

We mainly focus on the time complexity in learning  $\mathbf{U}_t$  given  $\mathcal{F}_t$  and the parameters learned in  $t - 1$ , i.e., Lines 7 to 15. First, the cost of sampling  $\mathbf{U}_t^{(s)}$  and  $\mathbf{V}_t^{(s)}$  for  $L$  times is approximately  $\mathcal{O}(n_t \cdot d \cdot L)$ . In each iteration, the time complexity of updating  $\boldsymbol{\mu}_{ui,t}$  and  $\mathbf{s}_{ui,t}$  using Eq.(5.25) and Eq.(5.26) are both approximately  $\mathcal{O}\left(d \cdot L \cdot (|\mathcal{N}_{t,i}^+| + |\mathcal{N}_{t,i}^-| + |\mathcal{W}_{t,i}|)\right)$ . Similarly, the cost of updating  $\boldsymbol{\mu}_{vi,t}$  and  $\mathbf{s}_{vi,t}$  are both approximately  $\mathcal{O}(d \cdot L \cdot |\mathcal{W}_{t,i}|)$ . Thus, the cost of updating all the parameters in one epoch is  $\mathcal{O}\left(d \cdot L \cdot (|\mathcal{N}_t^+| + |\mathcal{N}_t^-| + |\mathcal{W}_t|)\right)$ . Therefore, the total time complexity of learning  $\{\boldsymbol{\mu}_{ui,t}, \mathbf{s}_{ui,t}, \boldsymbol{\mu}_{vi,t}, \mathbf{s}_{vi,t}, i = 1, \dots, n_t\}$  is approximately  $\mathcal{O}\left(p \cdot d \cdot L \cdot (|\mathcal{N}_t^+| + |\mathcal{N}_t^-| + |\mathcal{W}_t| + n_t)\right)$ , where  $p$  is the number of iterations.

## 5.4 Experiments

In this section, we conduct experiments to evaluate the effectiveness of the proposed framework DNE. Specifically, we begin by introducing the datasets and the compared representative and state-of-the-art static and dynamic network embedding methods. We then compare DNE with other methods on next timestamp link prediction, which can indicate if DNE can capture the dynamic patterns of the network.

---

**Algorithm 3** Dynamic Network Embedding

---

**Require:**  $\mathcal{G}_{1:T}, \Delta\mathcal{G}_{1:T}, r, d, \sigma_0, D, L$ **Ensure:**  $\mathbf{U}_{1:T}, \mathbf{V}_{1:T}$ 

```
1: for  $t=1 : T$  do
2:   Construct  $\mathcal{F}_t$  from  $\Delta\mathcal{G}_t$  and  $\mathcal{G}_t$ 
3:   for  $i \in \mathcal{U}_t$  do
4:     Initialize  $\boldsymbol{\mu}_{ui,t}, \mathbf{s}_{ui,t}, \boldsymbol{\mu}_{vi,t}$ , and  $\mathbf{s}_{vi,t}$ 
5:     Calculate  $\tilde{\boldsymbol{\mu}}_{ui,t}, \tilde{\mathbf{s}}_{ui,t}, \tilde{\boldsymbol{\mu}}_{vi,t}$ , and  $\tilde{\mathbf{s}}_{vi,t}$  via Eq.(5.22)
6:   end for
7:   repeat
8:     Sample  $\mathbf{U}_t^{(s)}, \mathbf{V}_t^{(s)}, s = 1, \dots, L$ , using reparameterization
9:     for  $u_i \in \mathcal{U}_t$  do
10:      Update  $\boldsymbol{\mu}_{ui,t}$  as  $\boldsymbol{\mu}_{ui,t} = \boldsymbol{\mu}_{ui,t} - \eta \frac{\partial \tilde{\mathcal{L}}_t}{\partial \boldsymbol{\mu}_{ui,t}}$  via Eq.(5.25)
11:      Update  $\boldsymbol{\Sigma}_{ui,t}$  as  $\boldsymbol{\Sigma}_{ui,t} = \boldsymbol{\Sigma}_{ui,t} - \eta \frac{\partial \tilde{\mathcal{L}}_t}{\partial \boldsymbol{\Sigma}_{ui,t}}$  via Eq.(5.26)
12:      Update  $\boldsymbol{\mu}_{vi,t}$  as  $\boldsymbol{\mu}_{vi,t} = \boldsymbol{\mu}_{vi,t} - \eta \frac{\partial \tilde{\mathcal{L}}_t}{\partial \boldsymbol{\mu}_{vi,t}}$  via Eq.(5.27)
13:      Update  $\boldsymbol{\Sigma}_{vi,t}$  as  $\boldsymbol{\Sigma}_{vi,t} = \boldsymbol{\Sigma}_{vi,t} - \eta \frac{\partial \tilde{\mathcal{L}}_t}{\partial \boldsymbol{\Sigma}_{vi,t}}$  via Eq.(5.28)
14:     end for
15:   until convergence
16: end for
17: return  $\mathbf{U}_{1:T}, \mathbf{V}_{1:T}$ 
```

---

We also report the running time to show the efficiency of DNE. To further check the quality of the network embedding, we then compare DNE with other methods on node classification. Finally, we give a case study of DNE on DBLP to qualitatively check if DNE captures the concept drift of nodes. Further experiments are conducted to investigate the sensitivity of DNE on hyper-parameters.

### 5.4.1 Datasets

We conduct experiments on 5 publicly available dynamic network datasets, which includes three datasets from KONECT network collection [44]<sup>3</sup>, i.e., Infectious, Digg and DBLP, and two Google Plus datasets, i.e., Google Plus (Major) and Google Plus (Places).



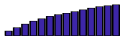


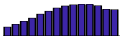




- **Infectious** [38]: It contains daily dynamic contact networks collected during the Infectious exhibition, where nodes represent exhibition visitors and edges represent face-to-face contacts.
- **Digg**: This dataset is the reply network of the news aggregation website *digg.com*, where each node is a user and each edge denotes the reply between two users.
- **DBLP**: This is the collaboration graph of authors from DBLP computer science bibliography. We choose snapshots from year 2000 to year 2013. The original networks contain millions of users. We filtered out authors who have few coauthors.
- **GPM**: This is a subset extracted from the Google+ social network dataset from [25]. The original dataset consists of 4 Google+ snapshots, where each snapshot includes both social links and node attributes. We use the attribute *Major* to select a subset of nodes and their neighbors as the GPM dataset. The major of the user is also used as the label for node classification. We selected 18 popular majors such as “Computer Science”, “Mechanical Engineering”, “Electrical Engineering” and “Political Science”.
- **GPP**: This is also a subset extracted from the Google+ social network from [25].

In this dataset, we use the attribute *Place Lives* to construct the dataset. Sim-

---

<sup>3</sup>Available at <http://konect.uni-koblenz.de/networks/>



Dataset	Nodes	Links	$T$	Cum. Node Dist.	New Link Dist.
Infectious	410	2,765	8		
Digg	30,398	86,404	14		
DBLP	113,168	3,737,308	14		
GPM	130,182	4,414,894	4		
GPP	149,671	5,531,455	4		

**Table 5.1:** Statistics of the Datasets

ilarly, the place a user lives is used as the node label. We selected 27 popular places such as “Chicago”, “London”, “San Francisco” and “Los Angeles”.

All the five datasets are used for next timestamp link prediction. In addition, GPM and GPP are also used for node classification as we have node labels. The statistics of the datasets are summarized in Table 5.1, where the cumulative user distribution and new link distribution in each timestamp are also given. We include dynamic networks of different sparsity, different cumulative user distributions and different new link distribution to give a comprehensive understanding of how DNE performs under various conditions.

#### 5.4.2 Compared Network Embedding Algorithms

We compare DNE with representative and state-of-the-art static and dynamic network embedding algorithms. The details are listed as follows:

- CN: Common neighbor [51] is a popular link prediction method. For any pair of nodes  $u_i$  and  $u_j$ , the link prediction strategy is to define the  $score(u_i, u_j) = |\mathcal{N}_{u_i} \cap \mathcal{N}_{u_j}|$ . We use it as a baseline to understand what’s the performance of utilizing the network structure without feature learning.

- MF: Matrix factorization [55] is a classical and popular network embedding algorithm on static networks. It factorizes the network into two low-rank matrices.
- CPTM: CP tensor-model [20] treat the dynamic network as a 3-dimensional tensor and uses CP decomposition to decompose the tensor to learn the representation of nodes. It’s a dynamic network embedding algorithm that considers the temporal patterns of networks.
- node2vec: node2vec [28] is the state-of-the-art static network embedding algorithm that learns low-dimensional vector representations of nodes that maximize the likelihood of preserving network neighborhoods of nodes.
- TMF: Temporal matrix factorization [97] is a variant of matrix factorization model proposed for dynamic networks. It extracts a low rank representation of the underlying adjacency matrices, in a way which are parameterized with time.
- LIST: LIST [98] characterizes the network dynamics as a function of time, which integrates the spatial topology of network at each timestamp and the temporal network evolution. It is state-of-the-art method for dynamic network embedding.

#### 5.4.3 Next Timestamp Link Prediction

In this section, we evaluate the next timestamp link prediction for each method, which aims at checking if the embedding can capture dynamic patterns. We use the first  $(T - 1)$  timestamps as training set, and the  $T$ -th timestamp as test set. We vary  $T$  as  $\{2, 3, 4, 5, 6, 7, 8\}$  for Infectious,  $\{2, 4, 6, 8, 10, 12, 14\}$  for Digg and DBLP

and  $\{2, 3, 4\}$  for GPP and GPM. *The purpose of increasing  $T$  is to simulate the dynamic networks as at each timestamp new links and new users will be introduced.* The datasets are characterized by extreme imbalance: the number of edges known to be present is often significantly less than the number of edges known to be absent. Thus, following the common way to evaluate the link prediction problem, we use AUC instead of accuracy to assess the performance [55]. AUC [21] measures the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one; a higher AUC would indicate a better predictive performance, which implies the effectiveness of the embedding.

There are some hyper-parameters to be tuned for the compared methods and DNE. To determine the hyper-parameters, we use the first  $(T - 2)$  timestamps as the training set and the  $(T - 1)$ -th timestamp as the validation set to tune the parameters. With the chosen parameters, we then use the first  $(T - 1)$  timestamps,  $\mathcal{G}_{1:T-1}$ , to train the model and calculate AUC on the  $T$ -th timestamp. Specifically, for DNE, we empirically set  $\sigma_0^2 = 1$ ,  $D = 10^{-3}$ ,  $L = 5$ ,  $r = 5$ , and  $d = 20$  for Infectious,  $d = 50$  for Digg and  $d = 100$  for the other datasets. More details about the sensitivity of DNE to the parameters will be discussed in Section 5.4.6. For DNE, we predict the link score between  $u_i$  and  $u_j$  at time  $t$  as  $\mathbf{u}_{i,t-1}^T \mathbf{u}_{j,t-1}$ . For MF, CPTM, node2vec, TMF, LIST and DNE, the embedding is randomly initialized, which introduces randomness of the result. To alleviate this issue, each experiment is conducted 5 times and the averaged performance with standard deviation is reported in Table 5.2, 5.3, 5.4, 5.5 and 5.6. From the tables, we make the following observations:

- When  $T = 2$ , the problem reduces to static network embedding, i.e., we learn embedding using  $\mathcal{G}_1$  only. The performance of DNE is better than CPTM, TMF and LIST for  $T = 2$ , which shows the effectiveness of DNE by exploiting first and second order proximity.

- The methods CPTM, TMF and LIST are temporal models based on matrix factorization; while their performances are better than that of MF when  $T > 2$ . This demonstrates the importance of taking the temporal patterns into consideration as temporal pattern contains the concept drift of nodes, which is important signal for learning better representation.
- Though node2vec doesn't consider temporal information, generally, the performance of node2vec is better than TMF and comparable to LIST. This is because node2vec explicitly capture the K-hop node similarity; while TMF and LIST simply learn embedding to reconstruct the links.
- Generally, DNE outperforms all the compared methods when  $T > 2$ , which is because DNE captures the temporal patterns using a Markovian process and at the same time exploits the first and second order proximity.

**Running Time Comparison:** One important goal of dynamic network embedding is to efficiently update/learn node representations when new links/nodes are introduced to the network.

T	2	3	4	5	6	7	8
CN	0.625±0.000	0.669±0.000	0.711±0.000	0.657±0.000	0.628±0.000	0.738±0.000	0.694±0.000
MF	0.624±0.004	0.649±0.002	0.704±0.007	0.628±0.003	0.620±0.003	0.715±0.003	0.671±0.007
CPTM	0.624±0.002	0.659±0.004	0.708±0.001	0.655±0.005	0.638±0.010	0.780±0.007	0.762±0.013
node2vec	<b>0.651±0.001</b>	0.686±0.001	0.730±0.001	0.651±0.001	0.654±0.001	0.736±0.002	0.719±0.004
TMF	0.626±0.002	0.691±0.002	0.741±0.002	0.641±0.002	0.670±0.004	0.743±0.002	0.759±0.005
LIST	0.627±0.003	0.690±0.003	0.753±0.003	0.669±0.002	0.678±0.003	0.779±0.003	0.767±0.004
DNE	0.643±0.003	<b>0.694±0.004</b>	<b>0.766±0.003</b>	<b>0.680±0.002</b>	<b>0.690±0.002</b>	<b>0.791±0.003</b>	<b>0.790±0.004</b>

**Table 5.2:** Link Prediction Performance Comparison on Infectious in Terms of AUC.

T	2	4	6	8	10	12	14
CN	0.598±0.000	0.599±0.000	0.602±0.000	0.605±0.000	0.606±0.000	0.608±0.000	0.613±0.000
MF	0.620±0.006	0.637±0.003	0.659±0.002	0.670±0.003	0.677±0.006	0.671±0.004	0.702±0.006
CPTM	0.629±0.005	0.654±0.004	0.677±0.004	0.694±0.004	0.695±0.004	0.710±0.007	0.726±0.006
node2vec	<b>0.650±0.002</b>	0.653±0.002	0.673±0.004	0.687±0.002	0.692±0.002	0.700±0.002	0.716±0.001
TMF	0.626±0.007	0.656±0.001	0.668±0.001	0.692±0.005	0.703±0.002	0.714±0.004	0.729±0.002
LIST	0.6290±0.003	0.668±0.004	0.6799±0.003	0.704±0.003	0.712±0.007	0.726±0.002	0.733±0.003
DNE	0.641±0.006	<b>0.672±0.006</b>	<b>0.685±0.002</b>	<b>0.713±0.005</b>	<b>0.730±0.002</b>	<b>0.732±0.0068</b>	<b>0.744±0.003</b>

**Table 5.3:** Link Prediction Performance Comparison on Digg in Terms of AUC.

T	2	4	6	8	10	12	14
CN	0.613±0.000	0.692±0.000	0.739±0.0000	0.781±0.000	0.819±0.000	0.833±0.000	0.887±0.000
MF	0.652±0.003	0.716±0.003	0.745±0.004	0.792±0.002	0.840±0.004	0.851±0.005	0.905±0.004
CPTM	0.657±0.005	0.722±0.006	0.756±0.002	0.819±0.006	0.858±0.001	0.879±0.005	0.922±0.005
node2vec	<b>0.686±0.002</b>	0.773±0.002	0.818±0.002	0.869±0.005	0.877±0.001	0.906±0.002	0.944±0.003
TMF	0.659±0.001	0.760±0.004	0.787±0.005	0.852±0.004	0.869±0.004	0.913±0.004	0.939±0.003
LIST	0.660±0.005	0.770±0.005	0.806±0.001	0.873±0.003	0.876±0.002	0.924±0.004	0.945±0.005
DNE	0.680±0.004	<b>0.779±0.005</b>	<b>0.829±0.003</b>	<b>0.880±0.002</b>	<b>0.887±0.002</b>	<b>0.935±0.003</b>	<b>0.953±0.002</b>

**Table 5.4:** Link Prediction Performance Comparison on DBLP in Terms of AUC.

To show the time efficiency of DNE in updating/learning node representations in a dynamic environment, we plot the time cost of network representation learning for node2vec, LIST and DNE on Digg and DBLP in Figure 5.3. From the figure, we have the following observations: (i) The time cost of node2vec and LIST increases significantly as  $T$  increases. This is because as  $T$  increases, new users and new links

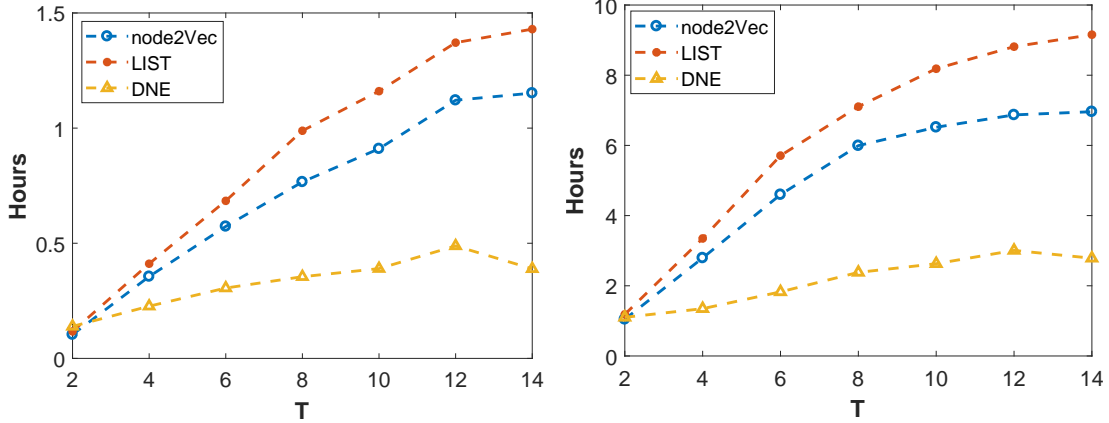
T	2	3	4
CN	0.7654±0.0000	0.8448±0.0000	0.8483±0.0000
MF	0.7822±0.0037	0.8669±0.0021	0.8549±0.0057
CPTM	0.7813±0.0043	0.8753±0.0032	0.8579±0.0039
node2vec	<b>0.8239</b> ±0.0028	0.8983±0.0028	0.8958±0.0016
TMF	0.7839±0.0055	0.8894±0.0057	0.8820±0.0037
LIST	0.7898±0.0021	0.9022±0.0015	0.8993±0.0028
DNE	0.8189±0.0031	<b>0.9107</b> ±0.0026	<b>0.9056</b> ±0.0021

**Table 5.5:** Link Prediction on GPM in Terms of AUC.

T	2	3	4
CN	0.7807±0.0000	0.8625±0.0000	0.8793±0.0000
MF	0.8038±0.0042	0.8796±0.0045	0.8837±0.0022
CPTM	0.8058±0.0031	0.8684±0.0022	0.8927±0.0024
node2vec	<b>0.8379</b> ±0.0036	0.9024±0.0028	0.9147±0.0020
TMF	0.8085±0.0063	0.8916±0.0038	0.9096±0.0024
LIST	0.8101±0.0043	0.9134±0.0024	0.9235±0.0036
DNE	0.8249±0.0047	<b>0.9202</b> ±0.0031	<b>0.9345</b> ±0.0029

**Table 5.6:** Link Prediction on GPP in Terms of AUC.

are created, which makes the network become larger. node2vec is a static network embedding algorithm, which requires retraining for each timestamp. Thus, the time cost of training node2vec increases as  $T$  increases. Though LIST is designed for capturing temporal patterns for dynamic networks, it assumes the network size is fixed in advance. In other words, it cannot dynamically update the embedding. At timestamp  $t$ , we need to retrain LIST using  $\mathcal{G}_{1:t}$ . Thus, the time cost of training LIST also increases as  $T$  increases; and (ii) The time cost of training DNE is much more efficient than node2vec and LIST, especially when the network becomes larger. This is because DNE can iteratively update or learn representation when new links and



(a) Time Cost on Digg

(b) Time Cost on DBLP

**Figure 5.3:** Running Time Comparison on Digg and DBLP

new users are introduced. At each time stamp, it’s time cost mainly relies on the size of the new users and links, and the size of the second order proximity extracted. These observations demonstrated the training efficiency of DNE for large dynamic networks.

In summary, it is worth noting that DNE can *significantly reduce the training time and slightly improve performance of next time-stamp link prediction*, which demonstrates the effectiveness of DNE in dynamically learning good network representation.

#### 5.4.4 Node Classification

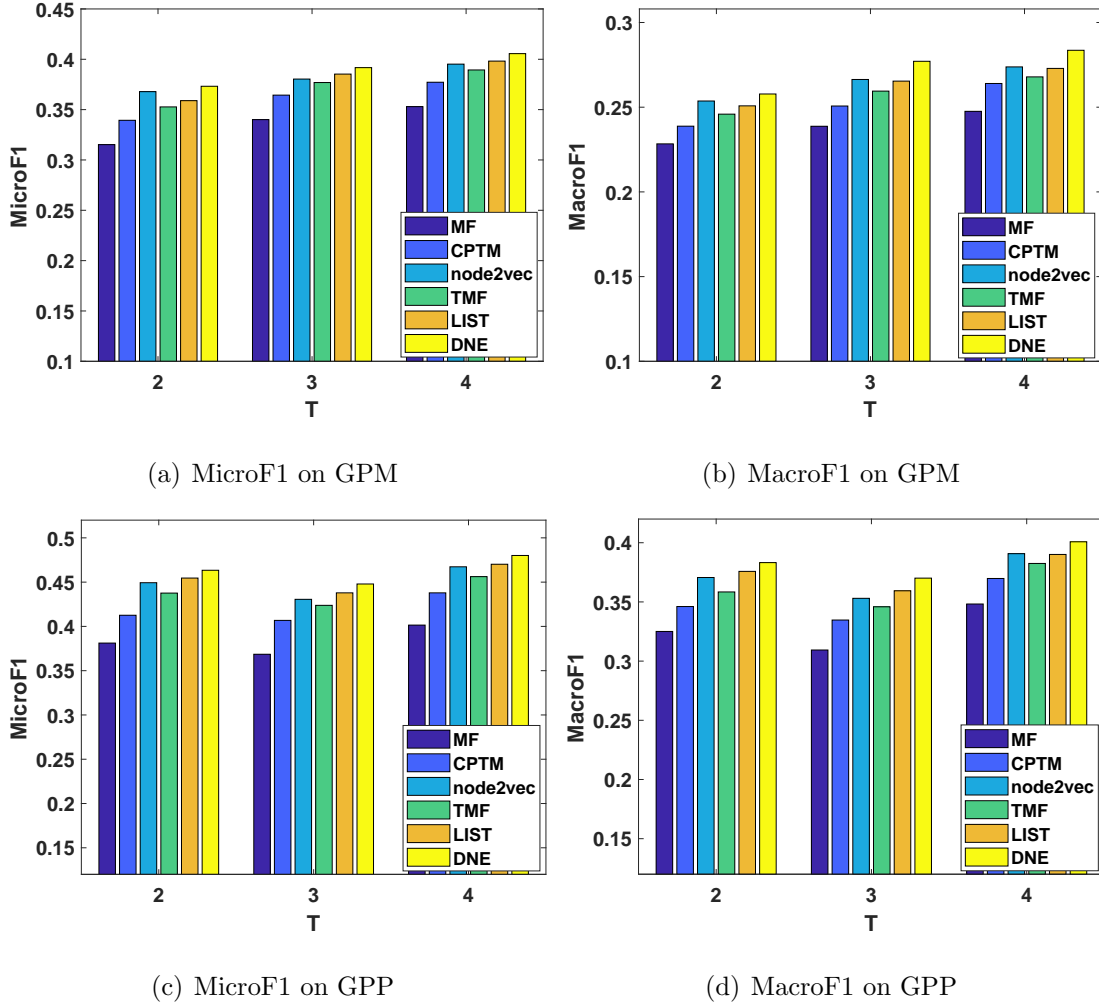
In this subsection, we conduct node classification to check the discriminativeness of representation learned by DNE. We use GPM and GPP for node classification as we have ground truth node labels for these two datasets. Note that the classes for these two datasets are imbalanced. Following the common way to measure the quality of classification on multi-class imbalanced datasets [92, 78], we adopt two widely used evaluation metrics, i.e., Micro-F1 and Macro-F1. The larger Micro-F1 and Macro-F1 scores are, the better the representation is for classification task. We compare with

the embedding algorithms described in Section 5.4.2 except CN as it is not suitable for node classification. Specifically, for both datasets, we use the first  $T$  timestamps,  $\mathcal{G}_{1:T}$ , to learn the network embeddings. We vary  $T$  as  $\{2, 3, 4\}$ . With the embeddings, we use the embedding at time  $T$ , i.e.,  $[\mathbf{U}_T, \mathbf{V}_T]$ , to predict the label of the nodes that join during  $(T - 1, T]$ . Specifically, we use nodes that already exist at time  $T - 1$ , i.e.,  $\mathcal{U}_{T-1}$ , to train a linear SVM. With the trained SVM, we predict the labels of node that join during  $(T - 1, T]$ , i.e.,  $\Delta\mathcal{U}_T$ . We design the experiment in this way to check the quality of the embedding, especially those of the new nodes. We use cross validation on the training data to tune the parameters. Each experiment is conducted 5 times and the average performances are shown in Figure 5.4. From the figure, we observe that: (i) Though CPTM, TMF and LIST are based on MF, they outperforms MF. This is because when new nodes join, they are more likely to build links with existing nodes that have similar labels with them. Thus, by capturing such temporal pattern, CPTM, TMF and LIST have better performance; (ii) node2vec outperforms CPTM and TMF, and is comparable to LIST. This is because the random walk sequences used by node2vec can provide better node similarity; and (iii) Generally, DNE outperforms the compared methods, which shows the ability of DNE in dealing with new nodes and new links by exploiting the diffusion process together with the first and second order proximity.

#### 5.4.5 *Discovering Temporal Concept Drift: A Case Study*

In this section, we further demonstrate that the proposed framework DNE can capture the temporal drift of networks with a case study on DBLP. We aim to use the embeddings of two consecutive years to detect authors whose coauthor list changed significantly in two consecutive years. The assumption is that good dynamic network embedding should capture the temporal drifting of network structures, and thus such





**Figure 5.4:** Node Classification Performance Comparison

coauthor (neighbor) changes should be reflected in the embedding. Specifically, we calculate  $\|\mu_t - \mu_{t-1}\|_2$  and sort the authors in descending order of  $\|\mu_t - \mu_{t-1}\|_2$ . A larger  $\|\mu_t - \mu_{t-1}\|_2$  means that there's significant change of user representation, which implies that the neighborhood of the user changes a lot. The top 5 authors with largest  $\|\mu_t - \mu_{t-1}\|_2$  for year 2002 to 2003 are reported in Table 5.7. For comparison, we also reported the number of unique coauthor differences for these authors between year 2002 and year 2003. From the table, we observe that: The retrieved authors all have a significant coauthor changes from year 2002 to year 2003, which shows that

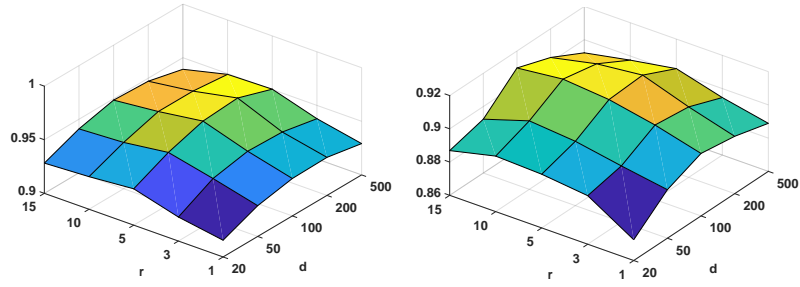
the embedding is able to capture such network structure changes. This case study further implies the effectiveness of DNE in learning representations that can capture temporal patterns.

#	$\ \boldsymbol{\mu}_t - \boldsymbol{\mu}_{t-1}\ _2$	Authors	Coauthor Diff
1	0.1047	HongJiang Zhang	152
2	0.0996	Steffen Staab	121
3	0.0979	Heung-Yeung Shum	102
4	0.0968	Erik D. Demaine	95
5	0.0896	Ian T. Foster	109

**Table 5.7:** Top 5 Authors Sorted by  $\|\boldsymbol{\mu}_t - \boldsymbol{\mu}_{t-1}\|_2$  for Year 2003

#### 5.4.6 Parameter Analysis

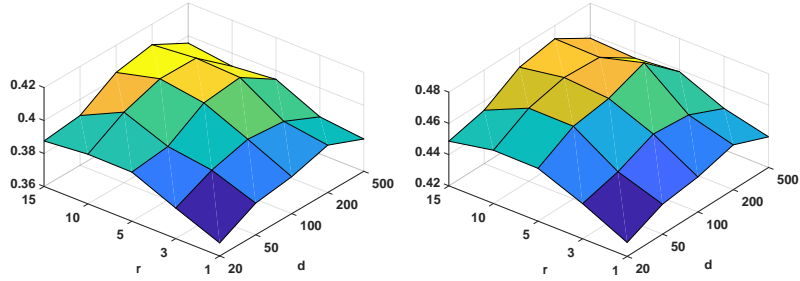
The proposed framework has two important parameters to be tuned, i.e.,  $r$  is the number of negative samples and  $d$  denotes the embedding dimension. In this section, we investigate the impact of these parameters on link prediction and node classification performance. We use the same experimental setting as previous section. For link prediction, we only show results in terms of AUC for  $T = 14$  on DBLP and  $T = 4$  on GPM as we have similar observations on other datasets and other values of  $T$ . Similarly, for node classification, we only show results in terms of MicroF1. We set  $\sigma_0^2 = 1$  and  $D = 10^{-3}$ . We vary the values of  $r$  as  $\{1, 3, 5, 10, 15\}$  and the values of  $d$  as  $\{20, 50, 100, 200, 500\}$ . Each experiment is conducted 5 times and the average results for link prediction and node classification are shown in Figure 5.5 and Figure 5.6, respectively. From these two figures, we observe that: (i) Generally, as the increase of  $d$ , the performance first increases and then decreases slightly after  $d$  reaches certain value. This is because embeddings with a small  $d$  don't have enough representation



(a) AUC on DBLP

(b) AUC on GPM

**Figure 5.5:** Parameter Analysis of DNE on Link Prediction



(a) MicroF1 on GPM

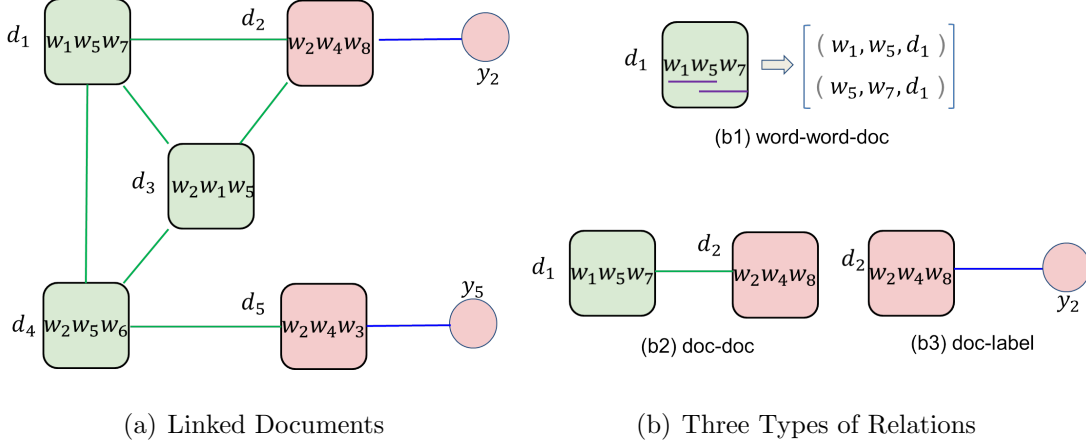
(b) MacroF1 on GPP

**Figure 5.6:** Parameter Analysis of DNE on Node Classification

capacity to capture the first and second order proximity; while embeddings with a too large  $d$  makes the model too complex, which can result in overfitting. A similar observation holds for  $r$ ; and (ii) The performance is generally better and more stable when the value of  $r$  is in  $[5, 10]$ , and the value of  $d$  is in  $[50, 200]$ . This observation eases the parameter selection.

## DOCUMENT NETWORK EMBEDDING

In attributed networks, attributes are represented as binary vectors. However, attributes can also present in more complex formats. For example, in social media, a user can have a short bio or description to introduce him/herself. The description can be treated as a document. Together with social links, it forms a document network (or linked documents). A toy example of a document network is illustrated in Figure 6.1(a) where  $\{d_1, d_2, \dots, d_5\}$  are documents and  $\{w_1, w_2, \dots, w_8\}$  are words in documents. In addition to content information, documents are linked and links suggest the inter-dependence of documents. One very straightforward way to handle a document network is to simply convert each document to bag-of-word representation, which essentially transform the document network to an attributed network and we can apply attribute network embedding algorithms to learn the representation. However, it ignores the word order and semantic meaning of words within a document, which can result in sub-optimal results. Therefore, we propose to investigate document network embedding that can simultaneously consider word semantic meanings and link information. In social media, it is possible to get certain user's label; while others may not reveal their labels. The partial label information in document networks can help to learn better representation for certain tasks such as node classification. Thus, we also take the partial label information into consideration



**Figure 6.1:** A Toy Example of Linked Documents.  $\{d_1, d_2, \dots, d_5\}$  Are Documents;  $\{w_1, w_2, \dots, w_8\}$  Are Words;  $y_2$  Is the Label of  $d_2$  and  $y_5$  Is the Label of  $d_5$ .

during representation learning. Next, we formally define the problem.

### 6.1 Problem Statement

Let  $\mathcal{D} = \{d_1, d_2, \dots, d_N\}$  be a set of  $N$  documents and  $\mathcal{W} = \{w_1, w_2, \dots, w_M\}$  be the word dictionary of size  $M$  for  $\mathcal{D}$ . Documents in  $\mathcal{D}$  are linked, which forms a document network  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , where each vertex is a document and  $e_{ij} = 1$  if documents  $d_i$  and  $d_j$  are connected. We use  $\mathcal{Y}$  to denote the subset of labeled documents in  $\mathcal{D}$  where  $y_i$  represents label information of the document  $d_i$ . Let  $\mathbf{D} \in \mathbb{R}^{d \times N}$  be the document embedding matrix where the  $i$ -th column of  $\mathbf{D}$ , i.e.,  $\mathbf{d}_i \in \mathbb{R}^{d \times 1}$ , is a  $d$ -dimensional vector representation of the document  $d_i$ . Similarly we use  $\mathbf{W} \in \mathbb{R}^{d \times M}$  to denote the word embedding matrix where the  $j$ -th column of  $\mathbf{W}$ , i.e.,  $\mathbf{w}_j \in \mathbb{R}^{d \times 1}$ , is a  $d$ -dimensional vector representation of the word  $w_j$  in  $\mathcal{W}$ .

With aforementioned definitions and notations, the problem under study is formally stated as:

*Given the document set  $\mathcal{D}$ , the document network  $\mathcal{G}$  and partial label information*

of  $\mathcal{D}$ , i.e.,  $\mathcal{Y}$ , we want to learn the document embedding matrix  $\mathbf{D}$  and the word embedding matrix  $\mathbf{W}$ . Mathematically, the problem is written as :

$$f(\mathcal{D}, \mathcal{G}, \mathcal{Y}) \rightarrow \{\mathbf{D}, \mathbf{W}\} \quad (6.1)$$

where  $f$  is the learning algorithm we propose to investigate.

## 6.2 The Proposed Framework

To model content, link and label information for word and document embedding, we extract three types of relations by examining Figure 6.1(a). The three types of relations are demonstrated in Figure 6.1(b): (1) word-word-document relations from content information shown in Figure 6.1(b1); (2) document-document relations from link information shown in Figure 6.1(b2); and (3) document-label relations from label information shown in Figure 6.1(b3). Next we elaborate these three relations and their corresponding model components before introducing the proposed framework LDE.

### 6.2.1 Modeling Word-Word-Document Relations

The distributional hypothesis that “you shall know a word by the company it keeps” suggests that a word has close relationships with its neighboring words. For example, the phrases *win the game* and *win the lottery* appear very frequently; thus the pair of words *win* and *game* and the pair of words *win* and *lottery* could have very close relationship. When we are only given the word *win*, we would highly expect the neighboring words to be words like *game* or *lottery* instead of words as *light* or *air*. This suggests that a good word representation should be useful for predicting its neighboring words, which is the essential idea of Skip-gram [57]. Meanwhile, depending on the topics of the documents, the probabilities of words appearing in the documents are different [10]. For example, though the appearance of the phrase *win*

*the lottery* is frequent, if we know that the topic of a document is about “sports”, we would expect words as *game* or *competition* after the word *win* instead of the word *lottery* because *win the game/competition* is more reasonable under the topic of “sports”. On the contrary, if the topic of the documents is about “lottery”, then we would expect *lottery* after *win*. These intuitions suggest that the predictions of neighboring words for a given word also strongly rely on the document. Therefore, we extract word-word-document relations from content information.

For a word  $w_i$ , we use a window of size  $c$  to extract  $w_i$  and its  $(c - 1)$  neighbors with  $w_i$  at the center and then  $w_i$  and each of its  $c - 1$  neighbors  $w_j$  form a pair as  $(w_i, w_j)$ . At the same time, we record which document the pair of words  $(w_i, w_j)$  comes from, say  $d_k$ . The pair of words  $(w_i, w_j)$  and the document  $d_k$  form a triplet  $(w_i, w_j, d_k)$ . An illustrative example of such process is given in Figure 6.1(b1), where window size  $c$  is 2. We denote all these triplets as a set  $\mathcal{P}$ . Note that in  $\mathcal{P}$ , there may be multiple  $(w_i, w_j, d_k)$  if the co-occurrence of  $w_i$  and  $w_j$  happens multiple times in  $d_k$  and there may be also  $(w_i, w_j, d_s)$  and  $(w_i, w_j, d_k)$  if the co-occurrence of  $w_i$  and  $w_j$  appears in both  $d_s$  and  $d_k$ . After extracting  $\mathcal{P}$ , the word-word-document relations can be captured by maximizing the average log probability:

$$\max_{\mathbf{W}, \mathbf{D}} \frac{1}{|\mathcal{P}|} \sum_{(w_i, w_j, d_k) \in \mathcal{P}} \log P(w_j | w_i, d_k) \quad (6.2)$$

where  $P(w_j | w_i, d_k)$  means the probability of given  $d_k$ , word  $w_j$  is a neighboring word of  $w_i$ , which is defined as

$$P(w_j | w_i, d_k) = \frac{\exp(\mathbf{w}_j^T \mathbf{w}_i + \mathbf{w}_j^T \mathbf{d}_k)}{\sum_{t=1}^M \exp(\mathbf{w}_t^T \mathbf{w}_i + \mathbf{w}_t^T \mathbf{d}_k)} \quad (6.3)$$

### 6.2.2 Modeling Document-Document Relations

Links between documents indicate the inter-dependence of documents. For example, a piece of online news about “sports” is likely to have hyperlinks to other news

on “sports” and a web mining article is likely to cite other web mining articles. Two linked documents are likely to share similar topics, which is a property commonly exploited in many tasks such as classification [69] and feature selection [75]. Therefore, we extract document-document relations from link information. For two linked document  $d_i$  and  $d_j$ , i.e.,  $e_{ij} = 1$ , the embedding vector for  $d_i$  is a good indicator of that of  $d_j$  since they are likely to share similar topics, which can be achieved by the following optimization problem:

$$\max_{\mathbf{D}} \frac{1}{|\mathcal{E}|} \sum_{i=1}^N \sum_{j:e_{ij}=1} \log P(d_j|d_i) \quad (6.4)$$

where  $|\mathcal{E}|$  is the number of links and  $P(d_j|d_i)$  is given as

$$P(d_j|d_i) = \frac{\exp(\mathbf{d}_j^T \mathbf{d}_i)}{\sum_{k=1}^N \exp(\mathbf{d}_k^T \mathbf{d}_i)} \quad (6.5)$$

From Eq.(6.5), we can see that if two linked documents have similar representations, then  $P(d_j|d_i)$  will be large. Thus, Eq.(6.4) aims at maximizing the similarity between two linked documents based on their embedding vectors.

### 6.2.3 Modeling Document-Label Relations

For the classification problem, we have some labeled samples and label information could guide the document embedding algorithms to learn better embedding. Let  $\mathbf{Y} \in \mathbb{R}^{d \times N_c}$  be the label embedding matrix where  $N_c$  is the number of unique labels and the  $k$ -th column of  $\mathbf{Y}$ ,  $\mathbf{y}_k$ , is the embedding vector for the  $k$ -th label.  $y_i$  is the label of the  $i$ -th document and the corresponding label embedding vector for  $y_i$  is  $\mathbf{y}_{y_i}$ . However, to avoid the notation confusion, we use  $\mathbf{y}_{d_i}$  instead of  $\mathbf{y}_{y_i}$  to denote the representation of the class label that is assigned to  $d_i$  in the remainder of the paper. A good document embedding vector for  $d_i$  should be a good indicator of the label of  $d_i$ . In other words, given the document, we should be able to predict its label; hence



we extract document-label relations from label information, which can be captured as follows:

$$\max_{\mathbf{Y}, \mathbf{D}} \frac{1}{|\mathcal{Y}|} \sum_{i: y_i \in \mathcal{Y}} \log P(y_{d_i} | d_i) \quad (6.6)$$

where  $P(y_{d_i} | d_i)$  is the probability that  $d_i$ 's label is  $y_{d_i}$ , which is given as

$$P(y_{d_i} | d_i) = \frac{\exp(\mathbf{y}_{d_i}^T \mathbf{d}_i)}{\sum_{k=1}^{N_c} \exp(\mathbf{y}_k^T \mathbf{d}_i)} \quad (6.7)$$

#### 6.2.4 Linked Document Embedding

With model components to capture content, link and label information, the proposed linked document embedding framework LDE is to solve the following optimization problem:

$$\begin{aligned} \min_{\mathbf{W}, \mathbf{D}, \mathbf{Y}} & -\frac{1}{|\mathcal{P}|} \sum_{(w_i, w_j, d_k) \in \mathcal{P}} \log P(w_j | w_i, d_k) - \frac{1}{|\mathcal{E}|} \sum_{i=1}^N \sum_{j: e_{ij}=1} \log P(d_j | d_i) \\ & - \frac{1}{|\mathcal{Y}|} \sum_{i: y_i \in \mathcal{Y}} \log P(y_i | d_i) + \gamma \Omega(\mathbf{W}, \mathbf{D}, \mathbf{Y}) \end{aligned} \quad (6.8)$$

In Eq.(6.8), the first term aims to learn document and word embeddings that are useful for predicting the neighbor word, which captures content information. The second term models link information and the third term captures label information that allows the document embeddings with the capability to predict labels.  $\Omega(\mathbf{W}, \mathbf{D}, \mathbf{Y})$  is the regularizer to prevent the model from overfitting. We can choose the regularizer with  $\ell_1$ -norm if the dimension of the embedding is high. Since we want to represent documents and words with low-dimensional vectors for classification, we use the Frobenius in our model:

$$\Omega(\mathbf{W}, \mathbf{D}, \mathbf{Y}) = \|\mathbf{W}\|_F^2 + \|\mathbf{D}\|_F^2 + \|\mathbf{Y}\|_F^2 \quad (6.9)$$

### 6.3 Learning LDE

In this section, we introduce the details of how to use stochastic gradient method to train the model. We will first introduce how to speed up the training process and

then give the update rules and the detailed algorithm.

### 6.3.1 Approximation by Negative Sampling

To update  $\mathbf{w}_j$ , we need to take derivative of  $\log P(w_j|w_i, d_k)$  w.r.t.  $\mathbf{w}_j$ , which is given by:

$$\nabla_{\mathbf{w}_j} \log P(w_j|w_i, d_k) = (1 + P(w_j|w_i, d_k))(\mathbf{w}_i + \mathbf{d}_k) \quad (6.10)$$

From Eq.(6.10), we find that updating  $\mathbf{w}_j$  requires the calculation of  $P(w_j|w_i, d_k)$ . However, the calculation of  $P(w_j|w_i, d_k)$  is expensive, because the denominator of  $P(w_j|w_i, d_k)$  is written as  $\sum_{t=1}^M \exp(\mathbf{w}_t^T \mathbf{w}_i + \mathbf{w}_t^T \mathbf{d}_k)$ . It requires summation over all the words, which could be very inefficient since the number of words is usually very large. To accelerate the speed, following the method used in Skip-gram model, we use the trick of negative sampling. In detail, the negative sampling is defined by the following objective function [57]:

$$\log \sigma(\mathbf{w}_j^T (\mathbf{w}_i + \mathbf{d}_k)) + \sum_{t=1}^K \mathbb{E}_{w_t \sim P_n(w)} [\log \sigma(-\mathbf{w}_t^T (\mathbf{w}_i + \mathbf{d}_k))] \quad (6.11)$$

which replaces every  $\log P(w_j|w_i, d_k)$  term in the objective function of Eq.(6.8). Thus the task becomes to distinguish the target word  $w_j$  from  $K$  words drawn from the noise distribution  $P_n(w)$ . The idea behind negative sampling is that we want to maximize the similarity between  $\mathbf{w}_j$  and  $(\mathbf{w}_i + \mathbf{d}_k)$  and minimize the similarity between a randomly sampled word  $\mathbf{w}_t$  and  $(\mathbf{w}_i + \mathbf{d}_k)$ . In this way, we can approximately maximize  $\log P(w_j|w_i, d_k)$ . In practice, the noise distribution is chosen to be  $U(w)^{3/4}/Z$ , where  $U(w)$  is the unigram distribution of the words and  $Z = \sum_w U(w)^{3/4}$  is the normalization term.

Thus, for a training instance  $(w_i, w_j, d_k) \in \mathcal{P}$ , we would draw  $K$  negative word samples, say one negative sample is  $w_t$ , from the noise distribution as  $w_t \sim P_n(w)$  and then we put  $(w_i, w_t, d_k)$  into the negative training set  $\mathcal{N}$ . It is easy to verify that

$|\mathcal{N}| = K|\mathcal{P}|$ . Now with  $\mathcal{N}$  and  $\mathcal{P}$ , we can approximate the first term in Eq.(6.8) using Eq.(6.11) as

$$\min_{\mathbf{W}, \mathbf{D}} - \frac{1}{|\mathcal{P}|} \sum_{(w_i, w_j, d_k) \in \mathcal{P}} \log \sigma(\mathbf{w}_j^T (\mathbf{w}_i + \mathbf{d}_k)) - \frac{1}{|\mathcal{P}|} \sum_{(w_i, w_t, d_k) \in \mathcal{N}} \log \sigma(-\mathbf{w}_t^T (\mathbf{w}_i + \mathbf{d}_k)) \quad (6.12)$$

Similarly, we approximate  $P(d_j|d_i)$  as

$$\log \sigma(\mathbf{d}_j^T \mathbf{d}_i) + \sum_{t=1}^K \mathbb{E}_{d_t \sim \tilde{P}_n(d)} \log \sigma(-\mathbf{d}_t^T \mathbf{d}_i) \quad (6.13)$$

where  $d_t$  in Eq.(6.13) is randomly sampled from documents that are not linked with  $d_i$ . Thus, for each linked document pair  $(d_i, d_j)$ , we need to randomly sample  $K$  documents,  $d_t$ , that are not linked to  $d_i$  and put  $(d_i, d_t)$  to the negative document set  $\mathcal{N}_E$ . We can see that  $|\mathcal{N}_E| = K|\mathcal{E}|$ . Now the second term in Eq.(6.8) can be written as

$$\min_{\mathbf{D}} - \frac{1}{|\mathcal{E}|} \left( \sum_{e_{ij} \in \mathcal{E}} \log \sigma(\mathbf{d}_j^T \mathbf{d}_i) + \sum_{(d_i, d_t) \in \mathcal{N}_E} \log \sigma(-\mathbf{d}_t^T \mathbf{d}_i) \right) \quad (6.14)$$

With the similar idea,  $P(y_{d_i}|d_i)$  can be approximated as

$$\log \sigma(\mathbf{y}_{d_i}^T \mathbf{d}_i) + \sum_{y \neq y_{d_i}} \log \sigma(-\mathbf{y}_y^T \mathbf{d}_i) \quad (6.15)$$

With these negative sampling approximations, the objective function of Eq.(6.8) can be approximated as

$$\begin{aligned} \min_{\mathbf{W}, \mathbf{D}, \mathbf{Y}} - \frac{1}{|\mathcal{P}|} \sum_{(w_i, w_j, d_k) \in \mathcal{P}} \log \sigma(\mathbf{w}_j^T (\mathbf{w}_i + \mathbf{d}_k)) - \frac{1}{|\mathcal{P}|} \sum_{(w_i, w_t, d_k) \in \mathcal{N}} \log \sigma(-\mathbf{w}_t^T (\mathbf{w}_i + \mathbf{d}_k)) \\ - \frac{1}{|\mathcal{E}|} \sum_{e_{ij} \in \mathcal{E}} \log \sigma(\mathbf{d}_j^T \mathbf{d}_i) - \frac{1}{|\mathcal{E}|} \sum_{(d_i, d_t) \in \mathcal{N}_E} \log \sigma(-\mathbf{d}_t^T \mathbf{d}_i) \\ - \frac{1}{|\mathcal{Y}|} \sum_{i: y_{d_i} \in \mathcal{Y}} [\log \sigma(\mathbf{y}_{d_i}^T \mathbf{d}_i) + \sum_{y \neq y_i} \log \sigma(-\mathbf{y}_y^T \mathbf{d}_i)] + \gamma \Omega(\mathbf{W}, \mathbf{D}, \mathbf{Y}) \end{aligned} \quad (6.16)$$

### 6.3.2 Updating Rules

We use stochastic gradient descent method to train the proposed model. Thus for each training sample, we need to update the involved word, document or label representations. For a given training instance,  $(w_i, w_j, d_k) \in \mathcal{P}$ , Eq.(6.16) reduces to

$$f_1 = -\frac{1}{|\mathcal{P}|} \log \sigma(\mathbf{w}_j^T(\mathbf{w}_i + \mathbf{d}_k)) + \gamma(\|\mathbf{w}_i\|_2^2 + \|\mathbf{w}_j\|_2^2 + \|\mathbf{d}_k\|_2^2)$$

The derivatives of the above equation w.r.t.  $\mathbf{w}_i$ ,  $\mathbf{w}_j$  and  $\mathbf{d}_k$  are given as

$$\begin{aligned} \nabla_{\mathbf{w}_i} f_1 &= \frac{1}{|\mathcal{P}|} [\sigma(\mathbf{w}_j^T(\mathbf{w}_i + \mathbf{d}_k)) - 1] \mathbf{w}_j + 2\gamma \mathbf{w}_i \\ \nabla_{\mathbf{d}_k} f_1 &= \frac{1}{|\mathcal{P}|} [\sigma(\mathbf{w}_j^T(\mathbf{w}_i + \mathbf{d}_k)) - 1] \mathbf{w}_j + 2\gamma \mathbf{d}_k \\ \nabla_{\mathbf{w}_j} f_1 &= \frac{1}{|\mathcal{P}|} [\sigma(\mathbf{w}_j^T(\mathbf{w}_i + \mathbf{d}_k)) - 1] (\mathbf{w}_i + \mathbf{d}_k) + 2\gamma \mathbf{w}_j \end{aligned} \quad (6.17)$$

Then  $\mathbf{w}_i$ ,  $\mathbf{w}_j$  and  $\mathbf{d}_k$  are updated as

$$\begin{aligned} \mathbf{w}_i &\leftarrow \mathbf{w}_i - \eta \nabla_{\mathbf{w}_i} f_1 \\ \mathbf{d}_k &\leftarrow \mathbf{d}_k - \eta \nabla_{\mathbf{d}_k} f_1 \\ \mathbf{w}_j &\leftarrow \mathbf{w}_j - \eta \nabla_{\mathbf{w}_j} f_1 \end{aligned} \quad (6.18)$$

where  $\eta$  is the learning rate.

Similarly, when the training instance is  $(w_i, w_t, d_k) \in \mathcal{N}$ , Eq.(6.16) is reduced to:

$$f_2 = -\frac{1}{|\mathcal{N}|} \log \sigma(-\mathbf{w}_t^T(\mathbf{w}_i + \mathbf{d}_k)) + \gamma(\|\mathbf{w}_i\|_2^2 + \|\mathbf{w}_t\|_2^2 + \|\mathbf{d}_k\|_2^2)$$

Then  $\mathbf{w}_i$ ,  $\mathbf{w}_t$  and  $\mathbf{d}_k$  are updated as

$$\begin{aligned} \mathbf{w}_i &\leftarrow \mathbf{w}_i - \eta \nabla_{\mathbf{w}_i} f_2 \\ \mathbf{d}_k &\leftarrow \mathbf{d}_k - \eta \nabla_{\mathbf{d}_k} f_2 \\ \mathbf{w}_t &\leftarrow \mathbf{w}_t - \eta \nabla_{\mathbf{w}_t} f_2 \end{aligned} \quad (6.19)$$

When a training instance is from  $\mathcal{E}$ , say  $(d_i, d_j)$ , we update  $\mathbf{d}_i$  and  $\mathbf{d}_j$  by the gradient descent method. When a training instance is from  $\mathcal{N}_E$ , say  $(d_i, d_t)$ , we

update  $\mathbf{d}_i$  and  $\mathbf{d}_t$ . Similarly, when the training instance is  $(d_i, y_i)$ , we update  $\mathbf{d}_i$  and  $\mathbf{Y}$  since all the label representation  $\mathbf{Y}$  is involved as shown in the fourth line of Eq.(6.16). We omit the detailed derivations here since they are very similar to the aforementioned ones.

### 6.3.3 Subsampling of Frequent Words

There is another issue we need to deal with. In large corpora, the most frequent words such as (“in”, “a”) can easily occur millions of times. In each epoch, these words will be trained millions of times correspondingly and the vector representations of these words will not change significantly after several epochs [57]. On the contrary, some rare words are trained less frequently in each epoch thus they need more epochs to train. To account the imbalance between rare and frequent words, we use the sub-sampling approach as in [57]: each word  $w_i$  in the training set is discarded with a probability computed by the formula:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \quad (6.20)$$

where  $f(w_i)$  is the frequency of the word  $w_i$  and  $t$  is a chosen threshold, typically around  $10^{-5}$ . The advantage of this subsampling formula is that it aggressively subsamples words whose frequencies are greater than  $t$  while preserving the ranking of the frequencies.

### 6.3.4 A Learning Algorithm for LDE

With the negative sampling and the update rules, the algorithm to learn LDE is summarized in Algorithm 4. We first prepare the training instances from line 1 to line 7. In line 8, we initialize the parameters  $\mathbf{W}, \mathbf{D}$  and  $\mathbf{Y}$ . Following the common practice, we initialize each element of  $\mathbf{W}, \mathbf{D}$  and  $\mathbf{Y}$  by randomly sampling from the

uniform distribution  $[-0.2, 0.2]$ . We then train LDE and update  $\mathbf{W}$ ,  $\mathbf{D}$  and  $\mathbf{Y}$  given the training data using the gradient descent method from line 9 to line 13. Finally, the document embedding  $\mathbf{D}$  and word embedding  $\mathbf{W}$  are obtained.

$\mathbf{D}$  is the document embedding which we name LDE-Doc. We can also represent documents using word embeddings. In particular, to get the document representation from word-embeddings  $\mathbf{W}$ , for a document  $d_i$ , we average all the words in the document as

$$\tilde{\mathbf{d}}_i = \frac{1}{N_i} \sum_{w_i \in d_i} \mathbf{w}_i \quad (6.21)$$

where,  $N_i$  is the length of the document  $d_i$  and  $\tilde{\mathbf{d}}_i$  is used as the document representation for  $d_i$ . We denote the document representation by word embedding as LDE-Word.

### 6.3.5 Time Complexity

When the training instance is  $(w_i, w_j, d_k) \in \mathcal{P}$ , from Eq.(6.17) and Eq.(6.18), we can see that the cost of calculating the derivative of  $f_1$  w.r.t.  $\mathbf{w}_i$  and updating  $\mathbf{w}_i$  are both  $\mathcal{O}(d)$ . With similar analysis, we find that the computational cost of calculating gradients and updating parameters are also  $\mathcal{O}(d)$  when training instances are from  $\mathcal{N}$ ,  $\mathcal{P}$ ,  $\mathcal{E}$ ,  $\mathcal{N}_E$  or  $\mathcal{Y}$ . Thus, we only need to count the size of the training data, which is  $(K+1)|\mathcal{P}| + (K+1)|\mathcal{E}| + N_c|\mathcal{Y}|$ . Therefore, the total computational cost in one epoch is  $((K+1)|\mathcal{P}| + (K+1)|\mathcal{E}| + N_c|\mathcal{Y}|)\mathcal{O}(d)$ . Considering the fact that  $\mathcal{E}$  is usually very sparse, the complexity is comparable to Skip-gram, which is scalable to millions of documents [57].

---

**Algorithm 4** LDE - Linked Document Embedding

---

**Require:**  $\mathcal{D}, \mathcal{G} = \{\mathcal{V}, \mathcal{E}\}, \mathcal{Y}, \lambda$ , window size  $c$ , dimension  $d$

**Ensure:**  $\mathbf{D}, \mathbf{W}$

- 1: Construct  $\mathcal{P}$  by using a sliding window size  $c$  to extract instances as  $(w_i, w_j, d_k)$  from documents where  $(w_i, w_j, d_k)$  is added to  $\mathcal{P}$  with the probability  $\sqrt{\frac{t}{f(w_i)}}$
  - 2: **for** each training sample in  $\mathcal{P}$  **do**
  - 3:     Draw  $K$  negative samples from noise distribution and put to  $\mathcal{N}$
  - 4: **end for**
  - 5: **for**  $e_{ij}$  in  $\mathcal{E}$  **do**
  - 6:     Randomly sample  $K$  documents that are not linked with  $d_i$  and put them into  $\mathcal{N}_E$
  - 7: **end for**
  - 8: Initialize  $\mathbf{W}, \mathbf{D}$  and  $\mathbf{Y}$
  - 9: **repeat**
  - 10:     **for** each training instance **do**
  - 11:         Update involved parameters using SGD as described in Section 6.3.2
  - 12:     **end for**
  - 13: **until** Convergence
  - 14: Return  $\mathbf{D}, \mathbf{W}$
- 

## 6.4 Experiments

In this section, we conduct experiments to evaluate the effectiveness of the proposed framework LDE. Specifically, we aim to answer the following questions:

- How effective is the proposed framework in learning document representations compared to the state-of-the-art methods?
- How does label information affect the performance of the proposed framework?

and

- Does the network information provide additional information for learning better document representations?

We begin by introducing the datasets and experimental settings, and then we compare LDE with the state-of-the-art algorithms for classification to answer the first question. We also investigate the sensitivity of LDE w.r.t. label and link information to answer the second and third questions.

#### 6.4.1 Datasets and Experimental Settings

### Datasets

The experiments are conducted on two real-world linked document datasets, DBLP and BlogCatalog. DBLP dataset is extracted by Arnetminer [80] from the DBLP website. Each document in DBLP dataset contains the title, authors and year of a paper. Some documents also contain venues, abstracts and reference papers. We use titles and abstracts as the document contents. Thus, we remove documents whose abstracts and titles are missing. We then choose six categories from the corpus<sup>1</sup>, including “Computer networks”, “Database:Data mining:Information retrieval”, “Computer graphics:Multimedia”, “Software engineering”, “Theoretical computer science” and “High-Performance Computing”. After that, we randomly select 2550 samples from each chosen category and add links between two documents if one document cites another document. BlogCatalog<sup>2</sup> is a blog directory where users can register their blogs under predefined categories. The categories are used as class labels of blogs. Each blog has a text description added by the owner, which is used as docu-

---

<sup>1</sup>Categories are defined according to venue by Arnetminer

<sup>2</sup><https://www.blogcatalog.com/>



Dataset	DBLP	BlogCatalog
# of documents	15,300	62,652
# of links	36,359	378,161
# of classes	6	27

**Table 6.1:** Statistics of the Datasets

ment content in our work. The homepage of each blog lists several blogs related to this blog, which forms links between a blog and its related blogs. In addition, if the owner of blog A follows the owner of blog B, we also add a link from blog A to blog B. We remove categories whose number of blogs are less than 500, which leaves us 27 categories and 62,652 blogs. Note that BlogCatalog dataset is unbalanced. For both datasets, we remove stop words and no further text normalizations such as stemming are done. The statistics of two datasets are summarized in Table 6.1.

## Evaluation Metrics

Our goal is to learn vector presentations of documents for classification. Therefore, we use classification performance to assess the quality of learned document representations. In fact, the classification task is also a common way to evaluate these word and document embedding algorithms with unsupervised settings [46]. Two widely used classification evaluation metrics, Micro-F1 and Macro-F1, are adopted. The larger the Micro-F1 and Macro-F1 scores are, the better the document representation is for the classification task.

### 6.4.2 Performance Comparison

To answer the first question, we compare the proposed framework LDE with other classical and state-of-the-art document representation learning algorithms. Since

Dataset	DBLP		BlogCatalog	
Name	Micro-F1	Macro-F1	Micro-F1	Macro-F1
BOW	78.50±0.64	78.61±0.63	46.35±0.42	40.78±0.43
RTM	74.05±0.68	74.08±0.71	44.62±0.35	39.60±0.37
Skip-gram	81.00±0.40	80.98±0.41	47.38±0.28	41.97±0.25
CBOV	77.33±0.73	77.31±0.73	45.43±0.44	39.03±0.29
PV-DM	84.25±0.26	84.25±0.26	48.35±0.24	42.78±0.23
PV-DBOW	80.81±0.30	80.82±0.29	47.56±0.23	41.68±0.25
LP	72.88±0.75	72.90±0.76	38.54±0.42	35.51±0.40
GC	84.75±0.82	84.74±0.81	48.76±0.37	42.98±0.34
TADW	85.59±0.65	85.58±0.64	49.85±0.31	43.95±0.32
CNN	84.07±0.45	84.09±0.48	49.01±0.51	43.38±0.47
PTE	85.26±0.47	85.23±0.49	50.36±0.43	44.58±0.42
LDE-Word	80.87±0.36	80.83±0.39	48.77±0.29	42.96±0.25
LDE-Doc	<b>87.69±0.42</b>	<b>87.70±0.45</b>	<b>53.14±0.42</b>	<b>46.85±0.39</b>

**Table 6.2:** Document Classification Performance Comparison on DBLP and BlogCatalog

LDA utilizes contents, links and labels during learning process, for fair comparison, the compared algorithms include state-of-the-art algorithms that utilizes links and contents such as RTM, TADW, contents and labels such as PTE, CNN and also graph-based classifier such as GC, which utilizes contents, link and labels for classification. The details of these algorithms are listed as follows:

- BOW [68]: the classical “bag-of-words” represent each document as a  $M$ -dimensional vector, where  $M$  is the size of the vocabulary and weight of each

dimension is calculated by the TFIDF scheme.

- RTM [13]: relational topic model is an extension of topic modeling that models document content and links between documents.
- Skip-gram [57]: one of the state-of-the-art word embedding model and its training objective is to find word representations that are useful for predicting the surrounding words of a selected word in a sentence. After obtaining word embeddings by Skip-gram, we use Eq.(6.21) to get document representations.
- CBOW [57]: another state-of-the-art word embedding model. Unlike Skip-gram, the training objective of CBOW is to find word representations that are useful for predicting the center word by its neighbors. Similarly, we use Eq.(6.21) to get document representations.
- PV-DM [46]: the distributed memory version of paragraph vector which considers the order of the words. It aims at learning document embeddings that are good at predicting the next given context.
- PV-DBOW [46]: the distributed bag-of-words version of paragraph vector model proposed in [46]. Unlike PV-DM, the word order is ignored in PV-DBOW. It aims to learn document representations that are good at predicting words in the document.
- LP [99]: a traditional semi-supervised algorithm based on label propagation, which performs classification by propagating label information from labeled data to unlabeled data through the graph. LP denotes a traditional method that utilizes both network information and label information for classification.

- GC [2] a graph-based classification method which utilizes both document contents, link and label information into a probabilistic framework for classification.
- CNN [41]: convolution neural network for classification. It uses word embeddings as input to train convolution neural network with label information<sup>3</sup>.
- TADW [96]: text-associated DeepWalk is a matrix factorization based method that utilizes both link and document data<sup>4</sup>.
- PTE [78]: predictive text embedding which considers label information to learn word embedding but cannot handle link information among documents.
- **LDE-Word**: the proposed framework trains both word embedding and document embedding. This variant uses the average of the word embeddings to represent a document.
- **LDE-Doc**: the proposed framework. Instead of using the word embeddings, we use the document embeddings directly as the representations of the documents.

The experiment consists of two phases, i.e., the representation learning phase and the document classification phase. During the representation learning phase, all the documents in the training set and testing set are used to learn word embeddings or document embeddings. For LDE-Word and LDE-Doc, labels of training data and link information are also used for learning embeddings during the representation learning phase. Labels of testing data are held out and no algorithm can use labels of testing data during the representation learning phase. During the classification phase, we use libsvm<sup>5</sup> [12] to train a SVM classifier using the learned document embeddings and

---

<sup>3</sup>Code available at [https://github.com/yoonkim/CNN\\_sentence](https://github.com/yoonkim/CNN_sentence)

<sup>4</sup>We use the code from <https://github.com/albertyang33/TADW>

<sup>5</sup>Available at <https://www.csie.ntu.edu.tw/~cjlin/libsvm/>

the training data. The trained SVM classifier is then assessed on the testing data<sup>6</sup>.

Dataset	Algorithm	Metrics	0%	20%	40%	60%	80%	100%
DBLP	LDE-Word	Micro-F1	76.62	77.30	78.25	78.79	79.69	80.87
		Macro-F1	76.63	77.26	78.23	78.76	79.65	80.83
	LDE-Doc	Micro-F1	78.68	79.05	81.85	83.87	85.65	87.69
		Macro-F1	78.68	79.04	81.85	83.87	85.65	87.70
BlogCatalog	LDE-Word	Micro-F1	45.45	45.92	46.36	47.05	47.98	48.77
		Macro-F1	39.62	40.09	40.73	41.47	42.20	42.96
	LDE-Doc	Micro-F1	46.57	47.12	48.83	50.17	51.62	53.14
		Macro-F1	40.65	41.24	42.76	44.15	45.51	46.85

**Table 6.3:** Effects of Label Density for LDE.

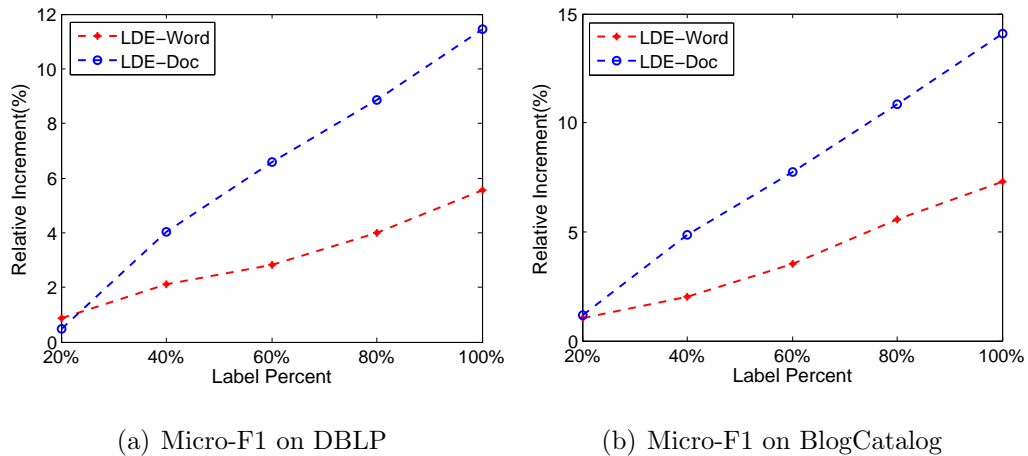
There are some parameters to set for the baseline algorithms. For a fair comparison, for Skip-gram, CBOW, PV-DM, PV-DBOW, CNN, RTM and LDE, we set the embedding dimension to be 100. For Skip-gram, CBOW, PV-DM, PV-DBOW and LDE, following the parameter setting suggestions in [57], we set the window size to be 7 and the number of negative samples also to be 7. We follow the setting in [78] for PTE and we use the default setting in the code of TADW. For the proposed model, we choose  $\gamma$  to be 0.0001. As of CNN, we use the default architecture in [41]. For both datasets, we randomly select 60% as training data and the remaining 40% as testing data. The random selection is conducted 5 times and the average micro-f1 and macro-f1 with standard deviations are reported in Table 6.2. From the table, we make the following observations:

<sup>6</sup>For Skip-gram, CBOW, PV-DM and PV-DBOW, we use the implementation by Gensim, which is available at <https://radimrehurek.com/gensim/>

- Skip-gram and PV-DM outperforms BOW slightly on both datasets. This shows that word/document embeddings can learn dense representations of documents which can improve the classification performance slightly. In contrast, LDE-Doc is much better than BOW on both datasets, which demonstrates the effectiveness of the proposed framework by incorporating link and label information.
- Most of the time, CNN outperforms other baseline methods. CNN uses label information for training and it is likely to obtain better performance. PTE outperforms CNN, which is consistent with previous observations [78].
- Comparing LDE-Doc and TADW, we can see that the performance of LDE-Doc is better than TADW. This is because though TADW utilizes link information, it doesn't consider label information for learning document representation.
- The performance of LDE-Doc is much better than LDE-Word. This is because LDE focuses on learning document representations. The link information and label information are used by LDE specific to document embedding instead of word embedding. LDE-Word is comparable to Skip-gram.
- The performance of LDE-Doc is better than the graph-based classification method GC, which also utilizes contents, link and label information for classification. This suggests that by utilizing the distributional hypothesis idea and exploiting the word-word-doc, doc-link and doc-label relationships, the learned document representations is good for classification.
- Though both PTE and LDE-Doc follow the idea of distributional hypothesis and use the label information, LDE-Doc significantly outperforms PTE. This is because in addition to label information, LDE-Doc also models the link information among documents which is pervasively available for linked documents.

- The proposed document embedding algorithm LDE-Doc outperforms representative document representation algorithms including PV-DM, RTM, PV-DBOW, PTE and TADW and graph-classification based methods such as GC, which further demonstrates that by considering link and label information, the proposed framework LDE is able to learn better document representations.

We conduct t-test on all performance comparisons and it is evident from t-test that all improvements are significant. In summary, the proposed framework can learn better document embeddings for classification by exploiting link and label information.



**Figure 6.2:** Relative Performance Improvement of LDE with Increasing Label Information.

### 6.4.3 Impact of Label Density

In this subsection, we perform experiments to investigate the effects of the label density on the quality of word and document embeddings. We first split each dataset into 60% and 40%, where 40% is fixed as testing data. From the 60% part, we sample  $x\%$  as labeled data in the embedding learning phase. The remaining  $(100-x)\%$  are not used for learning the embeddings. We vary  $x$  as  $\{0, 20, 40, 60, 80, 100\}$ . Similarly,

the experiment is composed of two phases. During the representation learning phase, we use all the documents with the  $x\%$  labeled data and link information to learn the embeddings. After the embeddings are learned, we use libsvm to train an SVM classifier. Each experiment is conducted 5 times and we report the average classification performance in terms of Micro-F1 and Macro-F1 in Table 6.3. To help understand the effects, we plot the relative performance improvement compared to that without label information ( $x = 0$ ) in terms of Micro-F1 in Figure 6.2. Note that we omit the figure for Macro-F1 since we have similar observations. From the table and the figures, we make the following observations:

- For both datasets, with the increase of labeled data in the representation learning phase, the performance of both word embedding and document embedding increases, which demonstrates that by incorporating label information, we can learn better document and word embeddings.
- From the figure, we can see that with the increase of labeled data, the difference between LDE-Word and LDE-Doc also increases, which indicates that label information is more useful for the proposed framework to learn document embeddings than word embeddings. This is reasonable because we explicitly model document and label relations to enable the capability of the learned representations in predicting labels.

#### 6.4.4 *Effects of Link Density*

In this subsection, we perform experiment to investigate the effects of the link density on the quality of word and document embeddings. Each time, we randomly sample  $x\%$  of links. We vary  $x$  as  $\{0, 20, 40, 60, 80, 100\}$ . We then split the dataset into 60% and 40%, where 60% are used for training and 40% are used for testing. The



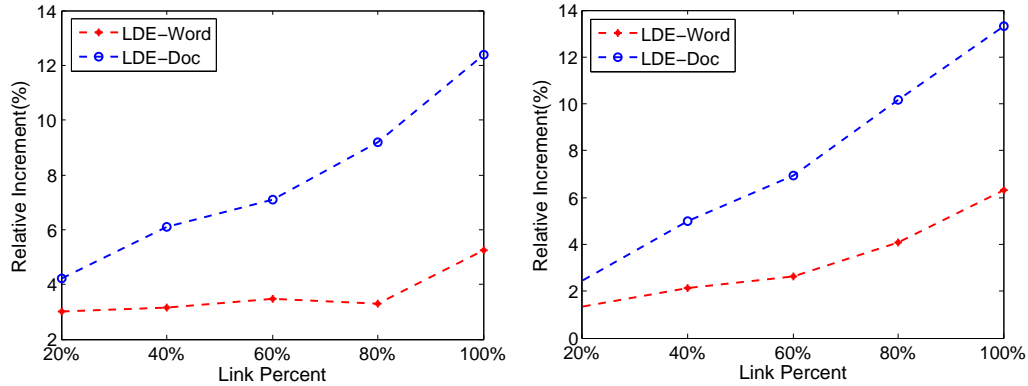
Dataset	Algorithm	Metrics	0%	20%	40%	60%	80%	100%
DBLP	LDE-Word	Micro-F1	76.83	79.15	79.25	79.51	79.35	80.87
		Macro-F1	76.82	79.12	79.23	79.48	79.32	80.83
	LDE-Doc	Micro-F1	78.03	81.31	82.78	83.56	85.20	87.69
		Macro-F1	78.05	81.30	82.79	83.57	85.22	87.70
BlogCatalog	LDE-Word	Micro-F1	45.87	46.48	46.84	47.07	47.75	48.77
		Macro-F1	39.87	40.38	40.97	41.52	42.13	42.96
	LDE-Doc	Micro-F1	46.89	48.04	49.23	50.15	51.65	53.14
		Macro-F1	40.91	41.62	42.65	44.27	45.71	46.85

**Table 6.4:** Effects of Link Density on LDE.

experiment is composed of two phases. During the representation learning phase, we use all the documents, the label for the training data and the  $x\%$  of links to learn embeddings. After learning embeddings, we use libsvm to train a SVM classifier. We report the classification performance in terms of Micro-F1 and Macro-F1 in Table 6.4. Similarly, to help understand the effects, we plot the relative performance improvement in terms of micro-f1 w.r.t. compared to that without links ( $x = 0$ ) in Figure 6.3. From the table and the figures, we make the following observations:

- For both datasets, as the percentage of links increases during the representation learning phase, the performance of both word embedding and document embedding increases, which demonstrates that by incorporating link information, we can learn better document and word embeddings.
- From the figure, we can see that as the percentage of links increases, the difference between LDE-Word and LDE-Doc also increases, which suggests that link information helps document embedding more than word embedding. The rea-

son is that we extract document and document relations from link information and then explicitly model them based on document embeddings.



(a) Micro-F1 on DBLP

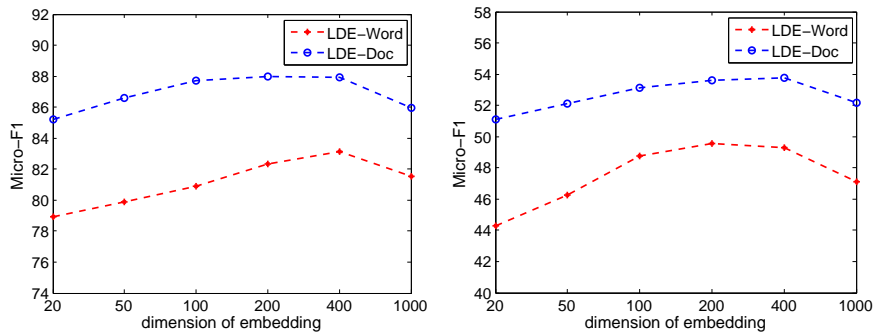
(b) Micro-F1 on BlogCatalog

**Figure 6.3:** Relative Performance Improvement of LDE with Increasing Link Information.

#### 6.4.5 Effects of Embedding Dimensions

In this subsection, we investigate how the embedding dimensions affect the performance of the proposed framework LDE. In detail, we first randomly select 60% as training and the remaining 40% as testing. All the documents, link information and label information of the 60% training data are used for learning document and word embeddings. After that, we train a SVM classifier to perform document classification with the learned document and word embeddings on the testing data. We vary the number of embedding dimension  $d$  as  $\{20, 50, 100, 200, 400, 1000\}$ . The random selection process is done 5 times and the average Micro-F1 are shown in Figure 6.4. Note that we only report Micro-F1 since the performance in terms of Macro-F1 is very close to Micro-F1. From the figures, we note that as the dimension of embeddings increases, the performance of both document embedding and word embedding first increases and then decreases. This is because when the embedding dimension is too small, the representation capability of the embedding vectors is not sufficient and

we may lose information. However, when the embedding dimension is too large, the model is too complex and we may overfit to the data.



(a) Micro-F1 on DBLP

(b) Micro-F1 on BlogCatalog

**Figure 6.4:** The Effects of Embedding Dimension on the Classification Performance of Document and Word Embedding.

## CONCLUSION AND FUTURE WORK

In this chapter, we summarize our research results and their broader impacts, and discuss promising research directions.

### 7.1 Summary

In this dissertation, we investigate network representation learning in social media. We provide principled approaches of exploiting social theories to design algorithms to tackle the challenges of representation learning for complex social networks. In particular, we study four innovative research tasks - (1) attributed network embedding; (2) signed network embedding; (3) dynamic network embedding; and (4) document network embedding. We propose novel frameworks to tackle the challenges of these networks and learn representations that not only capture the network structure but also the unique properties of these networks.

For attributed networks, we propose a novel RBM for linked data called pRBM, which is able to leverage both attribute and link information for representation learning. Specifically, pRBM is composed of a pair of RBMs so as to model nodes linked together. Gibbs sampling with mean-field inference is used to solve for efficient parameter estimation.

For signed networks, we introduce a new objective function for signed network embedding guided by extended structural balance theory and propose a deep learning framework SiNE to optimize this objective function. Via experiments on two signed networks in social media, we demonstrate that (1) the learned embedding can preserve the principle of signed social networks indicated by extended structural balance

theory; and (2) the embedding learned by SiNE can significantly improve the link prediction performance compared to representative baseline methods.

For dynamic networks, we investigate the challenging problem of dynamic network embedding where new nodes and new links can be introduced to dynamic networks anytime. We propose a novel probabilistic framework DNE, which exploits first-order and second-order proximity to learn better network embedding. In addition, DNE adopts a diffusion process to capture the node concept drift and can dynamically learn/update node representations for new nodes and new links. Experimental results on real-world datasets demonstrated the effectiveness of DNE in capturing the temporal patterns for next timestamp link prediction and node classification

For document networks, we investigate the problem of linked document embedding for classification. We propose a novel framework LDE that captures content, link and label information into a coherent model for learning document and word embeddings simultaneously. Experimental results on real-world datasets show that the proposed framework outperforms state-of-the-art document representation algorithms for classification.

Overall, we provide novel algorithms on four representative social networks, i.e., network with node attributes, network with edge attributes, networks with evolving nodes and links and networks with complex node information. These four novel algorithms facilitate the design of network representation on other types of social networks. In fact, many other types of social networks can be treated as the combination or variants of the studied social networks. For example, dynamic attributed network is a combination of attributed networks and dynamic networks. The principle or methodology used to design attributed network embedding and dynamic network embedding can be applied. Heterogeneous social networks, which is composed of multiple types of objects and various relations of between objects, can be treated as

a various of networks with edge attributes.

## 7.2 Future Work

Network representation learning is still in its early stages of development and remains an active area of exploration. Below we present some promising research directions:

- **Improving Scalability:** Though the majority of the network embedding algorithms are highly scalable in theory (i.e.,  $\mathcal{O}(E)$  training time), there is still significant work to be done in scaling node and graph embedding approaches to truly massive datasets (e.g., billions of nodes and edges) [30]. The massive datasets causes three problems: (i) large memory to fit in the training data; (ii) large storage to store a unique embedding for each individual node; and (iii) efficient methods to measure the similarity of node embeddings for link prediction. Therefore, it is necessary to develop representation learning frameworks that are truly scalable to realistic production settings. One potential way to solve these issues is to develop distributed network embedding algorithms, which distribute the burden of calculation and storage to many machines. Another promising direction is binary vector representation learning, which can alleviate the storage requirement and provide efficient approximate search [58].
- **Generalizing to Other Domains:** Though network embedding methods have achieved state-of-the-art performance on many network mining tasks, the majority of existing network embedding algorithms are designed for social networks or mainly evaluated with social networks. In real-world applications, there are many other domains which also require understanding and learning representations of networks, such as biology networks, airline networks and financial

networks. The principle used for learning social network representation may not be applicable to other domains. We need domain knowledge to help adapt network embedding algorithms on these domains.

- **Improving Interpretability:** One weakness of network embedding is its interpretability [30]. It is unclear if the embedding algorithms truly learn to represent relevant graph information, or just exploit statistical tendencies of benchmarks. In addition, a node is represented as a vector while we have no understanding of the semantic meanings each dimension carry. This is a big weakness compared with handcrafted features. Therefore, it is important to Improve the interpretability of network embedding algorithms.
- **Representing Networks as Vectors:** We have been focusing on learning *vector representations of nodes* in a network. There's another very important and promising network representation learning direction, which tries to learn *vector representations of networks*, i.e., representing each network as a vector. The learned network representation can be used for many network level downstream tasks such graph classification and measuring network similarity. In fact, many domains require vector representation of networks. For example, each chemical compound can be treated as a network, where an atom is a node in the network and the chemical bond between two atoms is the weighted edge in the network. With the learned vector representation of each chemical compound (network), we can train a classifier to predict if a chemical compound is toxic or not. The work on learning vector representations of networks is rather limited [60, 61]. Thus, it's a promising direction which needs further investigation.

## REFERENCES

- [1] C. Aggarwal and K. Subbian. Evolutionary network analysis: A survey. *ACM Computing Surveys (CSUR)*, 47(1):10, 2014.
- [2] R. Angelova and G. Weikum. Graph-based text classification: learn from your neighbors. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 485–492. ACM, 2006.
- [3] R. Bamler and S. Mandt. Dynamic word embeddings. In *International Conference on Machine Learning*, pages 380–389, 2017.
- [4] G. Beigi, J. Tang, S. Wang, and H. Liu. Exploiting emotional information for trust/distrust prediction. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 81–89. SIAM, 2016.
- [5] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591, 2002.
- [6] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828, 2013.
- [7] S. Bhagat, G. Cormode, and S. Muthukrishnan. Node classification in social networks. In *Social network data analytics*, pages 115–148. Springer, 2011.
- [8] G. Bishop and G. Welch. An introduction to the kalman filter. *Proc of SIGGRAPH, Course*, 8(27599-23175):41, 2001.
- [9] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, (just-accepted), 2017.
- [10] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.
- [11] D. Cartwright and F. Harary. Structural balance: a generalization of heider’s theory., 1956.
- [12] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27, 2011.
- [13] J. Chang and D. Blei. Relational topic models for document networks. In *Artificial Intelligence and Statistics*, pages 81–88, 2009.
- [14] K.-Y. Chiang, J. J. Whang, and I. S. Dhillon. Scalable clustering of signed networks using balance normalized cut. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 615–624. ACM, 2012.



- [15] M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. O. Wojtaszczyk. Sitting closer to friends than enemies, revisited. In *Mathematical Foundations of Computer Science 2012*, pages 296–307. Springer, 2012.
- [16] J. A. Davis. Clustering and structural balance in graphs. *Social networks. A developing paradigm*, pages 27–34, 1977.
- [17] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.
- [18] G. Dong and H. Liu. Feature engineering for machine learning and data analytics. 2018.
- [19] Y. Dong, J. Zhang, J. Tang, N. V. Chawla, and B. Wang. Coupledlp: Link prediction in coupled networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 199–208. ACM, 2015.
- [20] D. M. Dunlavy, T. G. Kolda, and E. Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(2):10, 2011.
- [21] T. Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.
- [22] A. Fischer and C. Igel. An introduction to restricted boltzmann machines. In *Iberoamerican Congress on Pattern Recognition*, pages 14–36. Springer, 2012.
- [23] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1243–1252, 2017.
- [24] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.
- [25] N. Z. Gong, W. Xu, L. Huang, P. Mittal, E. Stefanov, V. Sekar, and D. Song. Evolution of social-attribute networks: measurements, modeling, and implications using google+. In *Proceedings of the 2012 Internet Measurement Conference*, pages 131–144. ACM, 2012.
- [26] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [27] P. Goyal and E. Ferrara. Graph embedding techniques, applications, and performance: A survey. *arXiv preprint arXiv:1705.02801*, 2017.

- [28] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.
- [29] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1025–1035, 2017.
- [30] W. L. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 40(3):52–74, 2017.
- [31] A. Hassan, A. Abu-Jbara, and D. Radev. Extracting signed social networks from text. In *Workshop Proceedings of TextGraphs-7 on Graph-based Methods for Natural Language Processing*, pages 6–14. Association for Computational Linguistics, 2012.
- [32] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [33] F. Heider. Attitudes and cognitive organization. *The Journal of psychology*, 21(1):107–112, 1946.
- [34] G. Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.
- [35] G. E. Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.
- [36] G. E. Hinton and R. R. Salakhutdinov. Replicated softmax: an undirected topic model. In *Advances in neural information processing systems*, pages 1607–1614, 2009.
- [37] C.-J. Hsieh, K.-Y. Chiang, and I. S. Dhillon. Low rank modeling of signed networks. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 507–515. ACM, 2012.
- [38] L. Isella, J. Stehlé, A. Barrat, C. Cattuto, J.-F. Pinton, and W. Van den Broeck. What’s in a crowd? analysis of face-to-face behavioral networks. *Journal of theoretical biology*, 271(1):166–180, 2011.
- [39] I. Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.
- [40] M. Kim and J. Leskovec. Modeling social networks with node attributes using the multiplicative attribute graph model. *arXiv preprint arXiv:1106.5053*, 2011.
- [41] Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751, 2014.

- [42] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2013.
- [43] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
- [44] J. Kunegis. KONECT: the koblenz network collection. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013, Companion Volume*, pages 1343–1350, 2013.
- [45] J. Kunegis, S. Schmidt, A. Lommatzsch, J. Lerner, E. W. De Luca, and S. Albayrak. Spectral analysis of signed graphs for clustering, prediction and visualization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 559–570. SIAM, 2010.
- [46] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.
- [47] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.
- [48] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting positive and negative links in online social networks. In *Proceedings of the 19th international conference on World wide web*, pages 641–650. ACM, 2010.
- [49] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Signed networks in social media. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1361–1370. ACM, 2010.
- [50] K. Li, J. Gao, S. Guo, N. Du, X. Li, and A. Zhang. Lrbm: A restricted boltzmann machine based approach for representation learning on linked data. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 300–309. IEEE, 2014.
- [51] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *journal of the Association for Information Science and Technology*, 58(7):1019–1031, 2007.
- [52] D. Luo, C. H. Q. Ding, F. Nie, and H. Huang. Cauchy graph embedding. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 553–560, 2011.
- [53] S. Maniu, B. Cautis, and T. Abdesslem. Building a signed network from interactions in wikipedia. In *Databases and Social Networks*, pages 19–24. ACM, 2011.
- [54] M. McPherson, L. Smith-Lovin, and J. M. Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001.

- [55] A. K. Menon and C. Elkan. Link prediction via matrix factorization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 437–452. Springer, 2011.
- [56] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [57] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
- [58] V. Misra and S. Bhatia. Bernoulli embeddings for graphs. *arXiv preprint arXiv:1803.09211*, 2018.
- [59] K. P. Murphy. *Machine learning - a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, 2012.
- [60] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan. sub-graph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *arXiv preprint arXiv:1606.08928*, 2016.
- [61] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.
- [62] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114. ACM, 2016.
- [63] S. Papadopoulos, Y. Kompatsiaris, A. Vakali, and P. Spyridonos. Community detection in social media. *Data Mining and Knowledge Discovery*, 24(3):515–554, 2012.
- [64] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.
- [65] M. Qu, J. Tang, J. Shang, X. Ren, M. Zhang, and J. Han. An attention-based collaboration framework for multi-view network representation learning. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1767–1776. ACM, 2017.
- [66] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.
- [67] R. Salakhutdinov and G. E. Hinton. Deep boltzmann machines. In *AISTATS*, pages 448–455, 2009.
- [68] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

- [69] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.
- [70] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.
- [71] J. Tang, C. Aggarwal, and H. Liu. Node classification in signed social networks. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 54–62. SIAM, 2016.
- [72] J. Tang, C. Aggarwal, and H. Liu. Recommendations in signed social networks. In *Proceedings of the 25th International Conference on World Wide Web*, pages 31–40, 2016.
- [73] J. Tang, Y. Chang, C. Aggarwal, and H. Liu. A survey of signed network mining in social media. *ACM Computing Surveys (CSUR)*, 49(3):42, 2016.
- [74] J. Tang, X. Hu, and H. Liu. Is distrust the negation of trust?: the value of distrust in social media. In *Proceedings of the 25th ACM conference on Hypertext and social media*, pages 148–157. ACM, 2014.
- [75] J. Tang and H. Liu. Feature selection with linked data in social media. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 118–128. SIAM, 2012.
- [76] J. Tang and H. Liu. Unsupervised feature selection for linked social media data. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 904–912. ACM, 2012.
- [77] J. Tang, J. Liu, M. Zhang, and Q. Mei. Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th International Conference on World Wide Web*, pages 287–297. International World Wide Web Conferences Steering Committee, 2016.
- [78] J. Tang, M. Qu, and Q. Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1165–1174. ACM, 2015.
- [79] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077, 2015.
- [80] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 990–998. ACM, 2008.

- [81] L. Tang and H. Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 817–826. ACM, 2009.
- [82] T. Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071. ACM, 2008.
- [83] J. C. Turner. *Social influence*. Thomson Brooks/Cole Publishing Co, 1991.
- [84] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.
- [85] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6000–6010, 2017.
- [86] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *The Sixth International Conference on Learning Representations.*, 2018.
- [87] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.
- [88] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.
- [89] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016.
- [90] S. Wang, C. Aggarwal, J. Tang, and H. Liu. Attributed signed network embedding. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 137–146. ACM, 2017.
- [91] S. Wang, J. Tang, C. Aggarwal, Y. Chang, and H. Liu. Signed network embedding in social media. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 327–335. SIAM, 2017.
- [92] S. Wang, J. Tang, C. Aggarwal, and H. Liu. Linked document embedding for classification. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 115–124. ACM, 2016.

- [93] S. Wang, J. Tang, F. Morstatter, and H. Liu. Paired restricted boltzmann machine for linked data. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1753–1762. ACM, 2016.
- [94] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang. Community preserving network embedding. In *AAAI*, pages 203–209, 2017.
- [95] S. Wasserman. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.
- [96] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang. Network representation learning with rich text information. In *IJCAI*, pages 2111–2117, 2015.
- [97] W. Yu, C. C. Aggarwal, and W. Wang. Temporally factorized network modeling for evolutionary network analysis. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 455–464. ACM, 2017.
- [98] W. Yu, W. Cheng, C. C. Aggarwal, H. Chen, and W. Wang. Link prediction with spatial and temporal consistency in dynamic networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 3343–3349. AAAI Press, 2017.
- [99] X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002.

APPENDIX A  
OPTIMIZATION OF SINE



Following the common way, we employ the backpropagation to optimize the deep network for SiNE [47]. The key idea of backpropagation is to update the parameters in a backward direction by propagating "errors" backward to efficiently calculate the gradients. Basically, we want to optimize Eq. (4.3) w.r.t to  $\mathbf{X}$ ,  $\mathbf{x}_0$  and  $\theta$ . The key step of optimizing Eq. (4.3) is to get the gradient of  $\max(0, f(\mathbf{x}_i, \mathbf{x}_k) + \delta - f(\mathbf{x}_i, \mathbf{x}_j))$  and  $\max(0, f(\mathbf{x}_i, \mathbf{x}_0) + \delta - f(\mathbf{x}_i, \mathbf{x}_j))$  with respect to the parameters,  $\mathbf{X}$ ,  $\mathbf{x}_0$  and  $\theta$ . With the gradient, we then can update the parameters using gradient descent method. Let's first analyze  $\max(0, f(\mathbf{x}_i, \mathbf{x}_k) + \delta - f(\mathbf{x}_i, \mathbf{x}_j))$ .

- If  $\max(0, f(\mathbf{x}_i, \mathbf{x}_k) + \delta - f(\mathbf{x}_i, \mathbf{x}_j)) = 0$ , or equivalent,  $f(\mathbf{x}_i, \mathbf{x}_k) + \delta - f(\mathbf{x}_i, \mathbf{x}_j) \leq 0$ , the parameters have already been optimized for the inputs  $\mathbf{x}_i$  and  $\mathbf{x}_j$ . In other words, the gradient of  $\max(0, f(\mathbf{x}_i, \mathbf{x}_k) + \delta - f(\mathbf{x}_i, \mathbf{x}_j))$  is 0 when  $f(\mathbf{x}_i, \mathbf{x}_k) + \delta - f(\mathbf{x}_i, \mathbf{x}_j) \leq 0$ .
- If  $\max(0, f(\mathbf{x}_i, \mathbf{x}_k) + \delta - f(\mathbf{x}_i, \mathbf{x}_j)) > 0$ ,  $\max(0, f(\mathbf{x}_i, \mathbf{x}_k) + \delta - f(\mathbf{x}_i, \mathbf{x}_j))$  is equal to  $f(\mathbf{x}_i, \mathbf{x}_k) + \delta - f(\mathbf{x}_i, \mathbf{x}_j)$ .

The same idea can be applied to  $\max(0, f(\mathbf{x}_i, \mathbf{x}_0) + \delta - f(\mathbf{x}_i, \mathbf{x}_j))$ . Based on the aforementioned analysis, we only need to take gradient of  $f(\mathbf{x}_i, \mathbf{x}_j)$  w.r.t the parameters. Then we are able to get the gradient of Eq. (4.3) with some calculations. We will start from the parameters of the  $N$ -th layer and go backward to get derivatives for other layers. First, using Eq (4.8), the derivative of  $f(\mathbf{x}_i, \mathbf{x}_j)$  w.r.t  $\mathbf{w}$  is given as

$$\begin{aligned} \frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{w}} &= [1 - f^2(\mathbf{x}_i, \mathbf{x}_j)] \frac{\partial}{\partial \mathbf{w}} [\mathbf{w}^T \mathbf{z}^{N1} + b] \\ &= [1 - f^2(\mathbf{x}_i, \mathbf{x}_j)] \mathbf{z}^{N1} \end{aligned} \quad (\text{A.1})$$

and similarly, the derivative of  $f(\mathbf{x}_i, \mathbf{x}_j)$  w.r.t  $b$  is

$$\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial b} = 1 - f^2(\mathbf{x}_i, \mathbf{x}_j) \quad (\text{A.2})$$

Next, the gradient of  $f(\mathbf{x}_i, \mathbf{x}_j)$  w.r.t  $\mathbf{z}^{N1}$  is given as

$$\begin{aligned} \frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{z}^{N1}} &= [1 - f^2(\mathbf{x}_i, \mathbf{x}_j)] \frac{\partial}{\partial \mathbf{z}^{N1}} [\mathbf{w}^T \mathbf{z}^{N1} + b] \\ &= [1 - f^2(\mathbf{x}_i, \mathbf{x}_j)] \mathbf{w} \end{aligned} \quad (\text{A.3})$$

Let  $\boldsymbol{\delta}^{N1}$  be a vector with its  $s$ -th element  $\delta_s^{N1}$  defined as

$$\begin{aligned} \delta_s^{N1} &= \frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial z_s^{N1}} [1 - (z_s^{N1})^2] \\ &= [1 - (z_s^{N1})^2] [1 - f^2(\mathbf{x}_i, \mathbf{x}_j)] w_s \end{aligned} \quad (\text{A.4})$$

where  $z_s^{N1}$  is the  $s$ -th element of  $\mathbf{z}^{N1}$ .  $\boldsymbol{\delta}^{N1}$  is the "error" generated by the output layer and will propagate back to the  $N$ -th layer as shown later. Using the chain rule

and Eq. (4.7), the derivative of  $f(\mathbf{x}_i, \mathbf{x}_j)$  w.r.t.  $\mathbf{W}^N$  is given as:

$$\begin{aligned}\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{W}_{st}^N} &= \sum_k \frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial z_k^{N1}} \frac{\partial z_k^{N1}}{\partial \mathbf{W}_{st}^N} \\ &= \frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial z_s^{N1}} [1 - (z_s^{N1})^2] z_t^{(N-1)1}\end{aligned}\quad (\text{A.5})$$

With Eq. (A.4), the above equation is simplified as

$$\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{W}_{st}^N} = \delta_s^{N1} z_t^{(N-1)1} \quad (\text{A.6})$$

Similarly, the derivative of  $f(\mathbf{x}_i, \mathbf{x}_j)$  w.r.t.  $\mathbf{b}^N$  is given as:

$$\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial b_s^N} = \delta_s^{N1} \quad (\text{A.7})$$

From Eqs (A.6) and (A.7), we can see that the "error"  $\delta^{N1}$  is propagated backwards, i.e., it is used for the calculation of the gradients of the parameters for the  $N$ -th layer.

Generally, the "error" for the  $n$ -th layer is denoted as  $\delta^{n1}$ ,  $1 \leq n < N$ , with it's  $s$ -th element defined as

$$\delta_s^{n1} = \frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial z_s^{n1}} [1 - (z_s^{n1})^2] \quad (\text{A.8})$$

where the derivative of  $f(\mathbf{x}_i, \mathbf{x}_j)$  w.r.t  $\mathbf{z}^{n1}$  is given as

$$\begin{aligned}\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial z_s^{n1}} &= \sum_k \frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial z_k^{(n+1)1}} \frac{\partial z_k^{(n+1)1}}{\partial z_s^{n1}} \\ &= \sum_k \frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial z_k^{(n+1)1}} [1 - (z_k^{n1})^2] \mathbf{W}_{ks}^{n+1} \\ &= \sum_k \delta_k^{(n+1)1} \mathbf{W}_{ks}^{(n+1)}\end{aligned}\quad (\text{A.9})$$

Thus, we have

$$\delta_s^{n1} = [1 - (z_s^{n1})^2] \sum_k \delta_k^{(n+1)1} \mathbf{W}_{ks}^{(n+1)}, 1 \leq n < N \quad (\text{A.10})$$

It is clear from the above equation that the "error"  $\delta_k^{(n+1)1}$  from the  $(n+1)$ -th layer is back propagated to the  $n$ -th layer for the calculation of  $\delta_s^{n1}$ . With  $\delta_s^{n1}$ , the derivative of  $f(\mathbf{x}_i, \mathbf{x}_j)$  w.r.t  $\mathbf{W}^n$  and  $\mathbf{b}^n$  is given as

$$\begin{aligned}\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{W}_{st}^n} &= \sum_k \frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial z_k^{n1}} \frac{\partial z_k^{n1}}{\partial \mathbf{W}_{st}^n} \\ &= \frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial z_s^{n1}} [1 - (z_s^{n1})^2] z_t^{(n-1)1} \\ &= \delta_s^{n1} z_t^{(n-1)1}, 1 < n < N\end{aligned}\quad (\text{A.11})$$

and

$$\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial b_s^n} = \delta_s^{n1}, 1 \leq n < N \quad (\text{A.12})$$

The derivative of  $f(\mathbf{x}_i, \mathbf{x}_j)$  w.r.t  $\mathbf{W}^{11}$  and  $\mathbf{W}^{12}$  are

$$\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{W}^{11}} = \sum_k \frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial z_k^{11}} \frac{\partial z_k^{11}}{\partial \mathbf{W}^{11}} = \boldsymbol{\delta}^{11}(\mathbf{x}_i)^T \quad (\text{A.13})$$

and

$$\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{W}^{12}} = \boldsymbol{\delta}^{11}(\mathbf{x}_j)^T \quad (\text{A.14})$$

Finally, the derivative of  $f(\mathbf{x}_i, \mathbf{x}_j)$  w.r.t  $\mathbf{x}_i$  is

$$\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{x}_i} = \sum_k \frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial z_k^{11}} \frac{\partial z_k^{11}}{\partial \mathbf{x}_i} = (\mathbf{W}^{11})^T \boldsymbol{\delta}^{11} \quad (\text{A.15})$$

and the derivative of  $f(\mathbf{x}_i, \mathbf{x}_j)$  w.r.t  $\mathbf{x}_j$  is

$$\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{x}_j} = (\mathbf{W}^{12})^T \boldsymbol{\delta}^{11} \quad (\text{A.16})$$

Similarly, for  $f(\mathbf{x}_i, \mathbf{x}_k) = \tanh((\mathbf{w}^{N+1})^T \mathbf{z}^{N2} + \mathbf{b}^{N+1})$ , we define  $\delta_s^{n2}$  as

$$\delta_s^{n2} = \frac{\partial f(\mathbf{x}_i, \mathbf{x}_k)}{\partial z_s^{n2}} [1 - (z_s^{n2})^2] \quad (\text{A.17})$$

With the same procedure as  $f(\mathbf{x}_i, \mathbf{x}_j)$ , we can get the derivatives of  $f(\mathbf{x}_i, \mathbf{x}_k)$  w.r.t the parameters. We omit the details here and just

With these derivatives, it's easy to get the derivatives of the objective in Eq (4.3) w.r.t to the parameters. We denote the objective as  $\mathcal{L}(\mathbf{X}, \mathbf{x}_0, \theta)$ . In each iteration, the parameters are updated using gradient descent. Taking  $\mathbf{x}_0$  as an example, the update rule is given as

$$\mathbf{x}_0 \leftarrow \mathbf{x}_0 - \gamma \frac{\partial \mathcal{L}(\mathbf{X}, \mathbf{x}_0, \theta)}{\partial \mathbf{x}_0} \quad (\text{A.18})$$

where  $\gamma$  is the learning rate.

## A.1 Summary of Derivatives

In this section, we summarize the derivatives. For  $f(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\mathbf{w}^T \mathbf{z}^{N1} + b)$ , we have

$$\delta_s^{n1} = \begin{cases} [1 - (z_s^{n1})^2] \sum_k \delta_k^{(n+1)1} \mathbf{W}_{ks}^{(n+1)}, & 1 \leq n < N \\ w_s [1 - (z_s^{N1})^2] [1 - f^2(\mathbf{x}_i, \mathbf{x}_j)], & n = N \end{cases} \quad (\text{A.19})$$

and the derivatives of  $f(\mathbf{x}_i, \mathbf{x}_j)$  w.r.t  $\theta$  are given as

$$\begin{aligned}
\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{w}} &= [1 - f^2(\mathbf{x}_i, \mathbf{x}_j)] \mathbf{z}^{N1} \\
\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{W}_{st}^n} &= \delta_s^{n1} z_t^{(n-1)1}, 1 < n < N \\
\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{W}^{11}} &= \boldsymbol{\delta}^{11}(\mathbf{x}_i)^T \\
\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{W}^{12}} &= \boldsymbol{\delta}^{11}(\mathbf{x}_j)^T \\
\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial b} &= 1 - f^2(\mathbf{x}_i, \mathbf{x}_j) \\
\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial b_s^n} &= \delta_s^{n1}, 1 \leq n < N
\end{aligned} \tag{A.20}$$

the derivatives of  $f(\mathbf{x}_i, \mathbf{x}_j)$  w.r.t  $\mathbf{x}_i, \mathbf{x}_j$  are given as

$$\frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{x}_i} = (\mathbf{W}^{11})^T \boldsymbol{\delta}^{11} \frac{\partial f(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{x}_j} = (\mathbf{W}^{12})^T \boldsymbol{\delta}^{11} \tag{A.21}$$

For  $f(\mathbf{x}_i, \mathbf{x}_k) = \tanh(\mathbf{w}^T \mathbf{z}^{N2} + b)$ , we have

$$\delta_s^{n2} = \begin{cases} [1 - (z_s^{n2})^2] \sum_k \delta_k^{(n+1)2} \mathbf{W}_{ks}^{(n+1)}, & 1 \leq n < N \\ w_s [1 - (z_s^{N2})^2] [1 - f^2(\mathbf{x}_i, \mathbf{x}_k)], & n = N \end{cases} \tag{A.22}$$

and the derivatives of  $f(\mathbf{x}_i, \mathbf{x}_k)$  w.r.t  $\theta$  are given as

$$\begin{aligned}
\frac{\partial f(\mathbf{x}_i, \mathbf{x}_k)}{\partial \mathbf{w}} &= [1 - f^2(\mathbf{x}_i, \mathbf{x}_k)] \mathbf{z}^{N2} \\
\frac{\partial f(\mathbf{x}_i, \mathbf{x}_k)}{\partial \mathbf{W}_{st}^n} &= \delta_s^{n2} z_t^{(n-1)2}, 1 < n < N \\
\frac{\partial f(\mathbf{x}_i, \mathbf{x}_k)}{\partial \mathbf{W}^{11}} &= \boldsymbol{\delta}^{12}(\mathbf{x}_i)^T \\
\frac{\partial f(\mathbf{x}_i, \mathbf{x}_k)}{\partial \mathbf{W}^{12}} &= \boldsymbol{\delta}^{12}(\mathbf{x}_j)^T \\
\frac{\partial f(\mathbf{x}_i, \mathbf{x}_k)}{\partial b} &= 1 - f^2(\mathbf{x}_i, \mathbf{x}_k) \\
\frac{\partial f(\mathbf{x}_i, \mathbf{x}_k)}{\partial b_s^n} &= \delta_s^{n2}, 1 \leq n < N
\end{aligned} \tag{A.23}$$

the derivatives of  $f(\mathbf{x}_i, \mathbf{x}_k)$  w.r.t  $\mathbf{x}_i, \mathbf{x}_k$  are given as

$$\frac{\partial f(\mathbf{x}_i, \mathbf{x}_k)}{\partial \mathbf{x}_i} = (\mathbf{W}^{11})^T \boldsymbol{\delta}^{12} \quad \text{and} \quad \frac{\partial f(\mathbf{x}_i, \mathbf{x}_k)}{\partial \mathbf{x}_k} = (\mathbf{W}^{12})^T \boldsymbol{\delta}^{12} \tag{A.24}$$

## BIOGRAPHICAL SKETCH

Suhang Wang is a PhD candidate of Computer Science and Engineering at Arizona State University. He received his BS degree in Electrical and Computer Engineering from Shanghai Jiao Tong University, Shanghai, China, in 2012 and MS degree in Electrical Engineering:Systems from the University of Michigan, Ann Arbor, MI, in 2013. He was award multiple times of University Graduate Fellowships, and various Student Travel Awards and Scholarships. His research interests are in network representation learning, feature engineering, recommender systems and deep learning. He has published more than 35 innovative works in highly ranked journals and top conference proceedings such as IEEE TKDE, ACM TIST, Information Sciences, ACM SIGKDD, WWW, AAAI, IJCAI, CIKM, SDM, WSDM, ICDM and CVPR. He also worked as a research intern at IBM T.J. Watson Research Center in 2016 and a research data scientist intern in Clari in 2017. Updated information can be found at <http://www.public.asu.edu/~swang187/>.